UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA UNIVERSITARIA DE INGENIERÍA TÉCNICA DE TELECOMUNICACIÓN

PROYECTO FIN DE CARRERA

Virtualización de dispositivos para servicios en la Ciudad del Futuro

Autor
Néstor Lucas Martínez

Tutor
José Fernán Martínez Ortega
Dr. Ingeniero de Telecomunicación

Septiembre 2013

E.U.I.T.TELECOMUNICACIÓN

PROYECTO FIN DE CARRERA
PLAN 2000

TEMA: Tecnologías para la Ciudad del Futuro e Internet de las Cosas

TÍTULO: Virtualización de dispositivos para servicios en la Ciudad del Futuro

AUTOR: Néstor Lucas Martínez

TUTOR: Dr. José Fernán Martínez Ortega **Vº Bº.**

DEPARTAMENTO: DIATEL

Miembros del Tribunal Calificador:

PRESIDENTE: Cristina Bonis Téllez

VOCAL: José Fernán Martínez Ortega

VOCAL SECRETARIO: Lourdes López Santidrián

DIRECTOR:

Fecha de lectura:

Calificación: **El Secretario,**

RESUMEN DEL PROYECTO:

En esta memoria se da cuenta del trabajo de análisis, diseño, implementación y validación de una solución técnica en el marco de Internet de las Cosas y orientada a homogeneizar la diversidad de dispositivos que pueden operar en la Ciudad del Futuro.

Se parte de un estudio del estado del arte con el que se pretenden abordar los conocimientos necesarios para comprender los objetivos del proyecto. Tras este análisis se describe la arquitectura orientada a servicios nSOM, una capa de intermediación desarrollada en la EUITT y que opera en el ámbito de las redes con recursos limitados. Sobre esta arquitectura se propone un diseño para la virtualización de servicios que es finalmente validada en un escenario de prueba propio de la Ciudad del Futuro y de Internet de las Cosas.

Este Proyecto Fin de Carrera se enmarca dentro del Proyecto Europeo de Investigación *Web of Objects*, financiado por el subprograma de ayudas Avanza Competitividad I+D+I del Ministerio de Industria, Energía y Turismo (expediente TSI-020400-2011-29), y avalado por el programa de apoyo a la investigación ITEA2 (Information Technology for European Advancement) del organismo europeo EUREKA.

Science is the belief in the ignorance of experts.

Richard Feynman

Leave this world a little better than you found it.

Robert Baden Powell

A la memoria de mi abuelo Enrique Martínez Pereda, que falleció el 7 de septiembre de 2013.

Tu recuerdo siempre estará presente conmigo.

AGRADECIMIENTOS

Quisiera empezar agradeciendo a mis padres su constante apoyo. Fue mi madre quien me inculcó el valor y el gusto por el aprendizaje, y mi padre quien lo hizo por la tecnología y la ingeniería. A ellos les debo gran parte de lo que soy.

A Mónica, mi pareja, que siempre está ahí en los momentos buenos y en los malos. Su apoyo y su ánimo constante han hecho mucho más fácil el camino recorrido para llegar hasta aquí.

A José Antonio Sánchez, Pedro Castillejo, Jesús Rodríguez, Alexandra Cuerva, Huang Yuanjiang y al resto compañeros del Grupo de Investigación de Redes y Servicios de Próxima Generación. Siempre dispuestos a echar una mano, a compartir su experiencia y sobre todo a ayudar a sobre llevar los momentos más intensos de trabajo.

A María Jesús Martínez, compañera de trabajo en el proyecto Web of Objects. Nuestras largar conversaciones tras las reuniones de planificación y desarrollo han hecho posible este trabajo.

A Vicente Hernández, principal responsable de nuestra participación en el proyecto Web of Objects. Su manera de motivar a sacar lo mejor de cada uno y su trato entre iguales es algo que recordaré siempre.

Por último no puedo dejar de mencionar al Dr. José Fernán Martínez Ortega, tutor de este proyecto, y al que agradezco la confianza que ha depositado en mí, y la oportunidad que me ha dado para conocer de primera mano el trabajo de investigación que siempre he admirado, y en el que he querido desarrollar mi actividad desde que alcanza mi recuerdo.

RESUMEN

El aumento de las capacidades de interconexión de dispositivos de todo tipo está suponiendo una revolución en el campo de la prestación de servicios, tanto en la cantidad como en la variedad. Esta evolución ha puesto de manifiesto la necesidad de abordar un desarrollo tecnológico sin precedentes, donde la previsión de dispositivos interconectados e interoperando entre sí y con las personas alcanza cifras del orden de los millardos.

Esta idea de un mundo de cosas interconectadas ha dado lugar a una visión que se ha dado en llamar Internet de las Cosas. Un mundo donde las cosas de cualquier tipo pueden interactuar con otras cosas, incluyendo las que forman parte de redes con recurso limitados. Y esto además conduce a la creación de servicios compuestos que superan a la suma de las partes.

Además de la relevancia tecnológica, esta nueva visión enlaza con la de la Ciudad del Futuro. Un concepto que recurre a la convergencia de la energía, el transporte y las tecnologías de la información y las comunicaciones para definir una forma mediante la que lograr el crecimiento sostenible y competitivo, mejorando así la calidad de vida y abriendo el gobierno de las ciudades a la participación ciudadana.

En la línea de desarrollo que permite avanzar hacia la consecución de tales objetivos, este Proyecto Fin de Carrera propone una forma de virtualizar los servicios ofrecidos por la diversidad de dispositivos que van adquiriendo la capacidad de interoperar en una red. Para ello se apoya en el uso de una capa de intermediación orientada a servicios, nSOM, desarrollada en la EUITT. Sobre esta arquitectura se proponen como objetivos el diseño y desarrollo de una pasarela de servicios que haga accesibles desde la web los recursos ofrecidos en una red de sensores; el diseño y desarrollo de un registro de dispositivos y servicios en concordancia a la propuesta de arquitectura de referencia para Internet de las Cosas; y el estudio y diseño de un marco para la composición de servicios orquestados en redes de recursos limitados.

Para alcanzar estos objetivos primero se abordará un estudio del estado del arte donde se profundizará en el conocimiento de la las tecnologías para la interoperatividad entre cosas, abordando los principios de las redes inalámbricas de sensores y actuadores, las arquitecturas para las comunicaciones Máquina a Máquina e Internet de las Cosas, y la visión de la Web de las Cosas. Seguidamente se tratarán las tecnologías de red y de servicios de interés, para finalizar con un breve repaso a las tecnologías para la composición de servicios. Le seguirá una descripción detallada de la arquitectura nSOM y del diseño propuesto para este proyecto. Finalmente se propondrá un escenario sobre el que se llevarán a cabo diferentes pruebas de validación.

ABSTRACT

The increasing of the capabilities of all kind of devices is causing a revolution in the field of the provision of services, both in quantity and in diversity. This situation has highlighted the need to address unprecedented technological development, where the forecast of interconnected and interoperable devices between them and human beings reaches the order of billions. And these numbers go further when the connectivity of constrained networks is taken into account.

This idea of an interconnected world of things has led to a vision that has been called "The Internet of Things". It's a vision of a world where things of any kind can interact with other things, even those in the domain of a constrained network. This also leads to the creation of new composed services that exceed the sum of the parts.

Besides the technological interest, this new vision relates with the one from the Smart City. A concept that uses the convergence of the energy, the transport, and the information and communication technologies to define a way to achieve sustainable and competitive growth, improving the quality of life, and opening the governance of the cities to the participation.

In the development pathway to reach these goals, this Final Degree Dissertation proposes a way for the virtualization of the services offered by the variety of devices that are reaching the ability to interoperate in a network. For this it is supported by a service oriented middleware called nSOM that has been developed at EUITT. Using this architecture the goals proposed for this project are the design and development of a service gateway that makes available the resources of a sensor network through a web interface; the design and development of a Device & Service Registry according to the reference architecture proposal for the Internet of Things; and the study and design of a composition framework for orchestrated services in constrained networks.

To achieve these goals this dissertation begins with a State of the Art study where the background knowledge about the technologies in use for the interoperation of things will be settled. At first it starts talking about Wireless Sensor and Actuator Networks, the architectures for Machine-to-Machine communication and Internet of Things, and also the concepts for the Web of Things vision. Next the related network and services technologies are explored, ending with a brief review of service composition technologies. Then will follow a detailed description of the nSOM architecture, and also of the proposed design for this project. Finally a scenario will be proposed where a series of validation tests will be conducted.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ACRÓNIMOS

ACRÓNIMO	SIGNIFICADO
API	Application Programming Interface
ARM	Architectural Reference Model
BPEL	Business Process Execution Language
BPMN	Business Process Model and Notation
CORBA	Common Object Request Broker Architecture
DPWS	Device Profile for Web Services
DTD	Document Type Definition
EDA	Event Driven Architecture
EDGE	Enhanced Data rates for GSM Evolution
EIP	European Innovation Partnership
ESB	Enterprise Service Bus
ETSI	European Telecommunications Standards Institute
EU	European Union
EUITT	Escuela Universitaria de Ingeniería Técnica de Telecomunicación
eUTRAN	Evolved UTRAN
EXI	Efficient XML Interchange
GERAN	GSM EDGE Radio Access Network
GRyS	Grupo de Redes y Servicios de Próxima Generación
GSM	Global System for Mobile Communications
GUI	Graphical User Interface
GW	Gateway
HFC	Hybrid Fiber-Coaxial
HTTP	HyperText Transfer Protocol
IDL	Interface Definition Language
IEEE	Institute of Electrical and Electronics Engineering
IERC	IoT European Research Cluster
IETF	Internet Engineering Task Force
IoT	Internet of Things
IoT-A	Internet of Things – Architecture

IoT-GSI	Global Standards Initiative on Internet of Things
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
ISA	International Society of Automation
ISO	International Organization for Standardization
ITEA	Information Technology for European Advancement
ITU-T	International Telecommunications Union, Telecommunications Standardization Sector
JDBC	Java DataBase Connectivity
JMS	Java Message Service
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
LED	Light Emitting Diode
LoWPAN	Low power Wireless Personal Area Networks
LTE	Long Term Evolution
M2M	Machine-to-Machine
MAC	Medium Access Control
M-BUS	Meter Bus
mIa	M2M Application Interface
mId	M2M Device Interface
MIT	Massachusetts Institute of Technology
NFC	Near Field Communication
nSOF	nano Service Oriented Framework
nSOL	nano Semantic Ontology Language
nSOM	nano Service Oriented Middleware
NTC	Constrained Network
NTU	Unconstrained Network
OASIS	Organization for the Advancement of Structured Information Standards
OMG	Object Management Group
OSGi	Open Services Gateway Initiative
OSI	Open Systems Interconnection
PDU	Protocol Data Unit
PFC	Proyecto Fin de Carrera

PIB	Producto Interior Bruto
PLC	Power Line Communications
RDF	Resource Definition Framework
RFID	Radio Frequency IDentification
RMI	Remote Method Invocation
ROLL	Routing Over Low power and Lossy networks
SCC	Smart Cities and Communities
SETIS	Strategic Energy Technologies Information System
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SOC	Service Oriented Computing
SQL	Structured Query Language
SunSPOT	Sun Small Programmable Object Technology
TCP	Transport Control Protocol
TISPAN	Telecoms & Internet converged Services & Protocols for Advanced Networks
UDP	User Datagram Protocol
UPM	Universidad Politécnica de Madrid
UPnP	Universal Plug and Play
URI	Universal Resource Identifier
URL	Universal Resource Locator
USB	Universal Serial Bus
UTRAN	UMTS Terrestrial Radio Access Network
W3C	World Wide Web Consortium
WiMAX	Worldwide Interoperability for Microwave Access
WLAN	Wireless Local Area Networks
WoO	Web of Objects
WoT	Web of Things
WSAN	Wireless Sensor and Actuator Network
WSN	Wireless Sensor Network
xDSL	Digital Subscriber Line Technologies
XML	eXtensible Markup Language

Capítulo 1

Introducción y Objetivos

1.1. Introducción

1.1.1. La Ciudad del Futuro

La Ciudad del Futuro, expresión con la que en esta memoria se hará referencia al término *smart city*, corresponde a un concepto de desarrollo urbano sostenible y competitivo sobre el que lleva hablándose desde los últimos años de la década de los 90. Por aquél entonces la profusión de los servicios TIC (Tecnologías de la Información y la Comunicación), y en particular de Internet, dio como resultado que la definición de una Ciudad del Futuro estuviera estrechamente relacionada con el desarrollo y despliegue de las citadas tecnologías.

Durante los siguientes años varias ciudades de todo el mundo trataron de aplicar su propia interpretación de la Ciudad del Futuro en diferentes proyectos de desarrollo urbano llevados a cabo en colaboración con empresas, universidades y centros de investigación. Proyectos como la Ciudad del Futuro de San Diego (San Diego "*City of the Future*"), *Smart Capital* en Ottawa, *Intelligent Island* en Singapur dentro del plan IT2000 o los proyectos *Smart Santander* y *Smart City Málaga* en España son algunos ejemplos.

Todos estos proyectos tenían en común el objetivo de exponer diferentes puntos de vista sobre el concepto de una ciudad inteligente para la Ciudad del Futuro. En 2007 el Centro de Ciencia Regional de la Universidad Tecnológica de Viena condujo un proyecto en el que se analizaron varios de estos proyectos en la Unión Europea, logrando identificarse seis ejes o características principales para evaluar y comprender el desarrollo de una Ciudad del Futuro: *smart economy*, *smart mobility*, *smart environment*, *smart people*, *smart living* y *smart governance* [1].

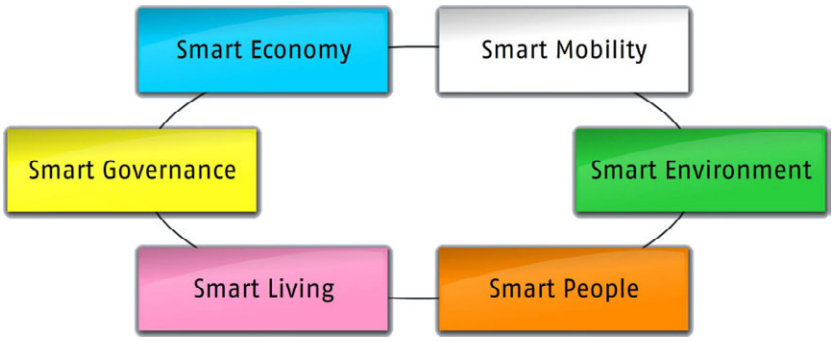

Figura 1. Los seis ejes de la Ciudad del Futuro [1]

El informe en el que se publicaron los resultados del citado proyecto describe aquellos factores que se consideran básicos como objetivos a cumplir por Ciudad del Futuro en cada una de las características enumeradas. Estos factores se muestran en la Tabla 1.

Características	Factores
Smart Economy (Competitividad)	Espíritu innovador
	Espíritu empresarial
	Imagen económica y de marca
	Productividad
	Flexibilidad del mercado laboral
	Arraigo internacional
	Capacidad de transformación
Smart People (Capital social y humano)	Nivel de cualificación
	Afinidad por el aprendizaje constante
	Pluralidad social y étnica
	Flexibilidad
	Creatividad
	Cosmopolitismo, mentalidad abierta
	Participación en la vida pública
Smart Governance (Participación)	Participación en la toma de decisiones
	Servicios públicos y sociales
	Gobierno transparente
	Perspectivas y estrategias políticas
Smart Mobility (Transporte y TIC)	Accesibilidad local
	Accesibilidad (inter-)nacional
	Disponibilidad de infraestructuras TIC
	Sistemas de transporte sostenibles, innovadores y seguros
Smart Environment (Recursos naturales)	Atractivo de las condiciones naturales
	Contaminación
	Protección del medio ambiente
	Gestión sostenible de los recursos
Smart Living (Calidad de vida)	Equipamientos culturales
	Condiciones de salud
	Seguridad individual
	Calidad de vivienda
	Servicios educativos
	Atracciones turísticas
	Cohesión social

Tabla 1. Características y factores de la Ciudad del Futuro

1.1.2. Europa 2020

El 3 de marzo de 2010 la Comisión de la Unión Europea publicó una estrategia para el crecimiento sostenible, inteligente e inclusivo [2]. Esta estrategia plantea alcanzar los siguientes objetivos de cara al año 2020:

- Incrementar la tasa de empleo de la población con edades comprendidas entre los 20 y los 64 años desde el 69% actual hasta al menos el 75%.
- Alcanzar el objetivo de invertir el 3% del PIB (Producto Interior Bruto) en I+D+i (Investigación, Desarrollo e innovación), en particular mediante la mejora de las condiciones para la inversión del sector privado en I+D+i, así como el desarrollo de nuevos indicadores para seguir las innovaciones.
- Reducir las emisiones de gas con efecto invernadero al menos en un 20% comparado a los niveles de 1990, o en un 30% si las condiciones lo permiten, incrementando la participación de la energía renovable en el consumo final en un 20%, y alcanzando una mejora de eficiencia energética en un 20%.
- Reducir el abandono escolar del 15% actual al 10%, e incrementar el porcentaje de población entre 30 y 34 años que hayan completado estudios superiores del 31% a al menos el 40%.
- Reducir el número de europeos que viven por debajo del umbral de pobreza en un 25%, sacando a 20 millones de personas de la misma.

Para alcanzar estos objetivos la Unión Europea ha puesto en marcha varias iniciativas, entre las que destaca en el ámbito de este proyecto la EIP (*European Innovation Partnership* o Colaboración Europea para la Innovación) en Ciudades y Comunidades Inteligentes (SCC o *Smart Cities and Communities*) [3]. Esta colaboración tiene como objetivo catalizar el progreso en áreas donde la producción, distribución y uso de energía, el transporte y la movilidad, y las tecnologías de la información y la comunicación se relacionan entre sí ofreciendo nuevas oportunidades multidisciplinares para mejorar los servicios a la par que se reducen los consumos de energía y recursos, así como la emisión de gases de efecto invernadero y otros contaminantes [4].

Fruto de esta colaboración es la Iniciativa Europea para las Ciudades del Futuro [5], que bajo el control de SETIS (*Strategic Energy Technologies Information System* o Sistema de Información de Tecnologías Estratégicas de la Energía) plantea una serie de objetivos estratégicos para demostrar la viabilidad de avanzar con rapidez hacia los objetivos energéticos y climáticos a nivel local mientras

que se prueba a los ciudadanos que su calidad de vida y su economía local puede verse mejorada a través de la inversión en eficiencia energética y reducción de las emisiones de carbono. Y todo ello implicando a las autoridades locales a través del Pacto de los Alcaldes [6] para multiplicar su impacto.

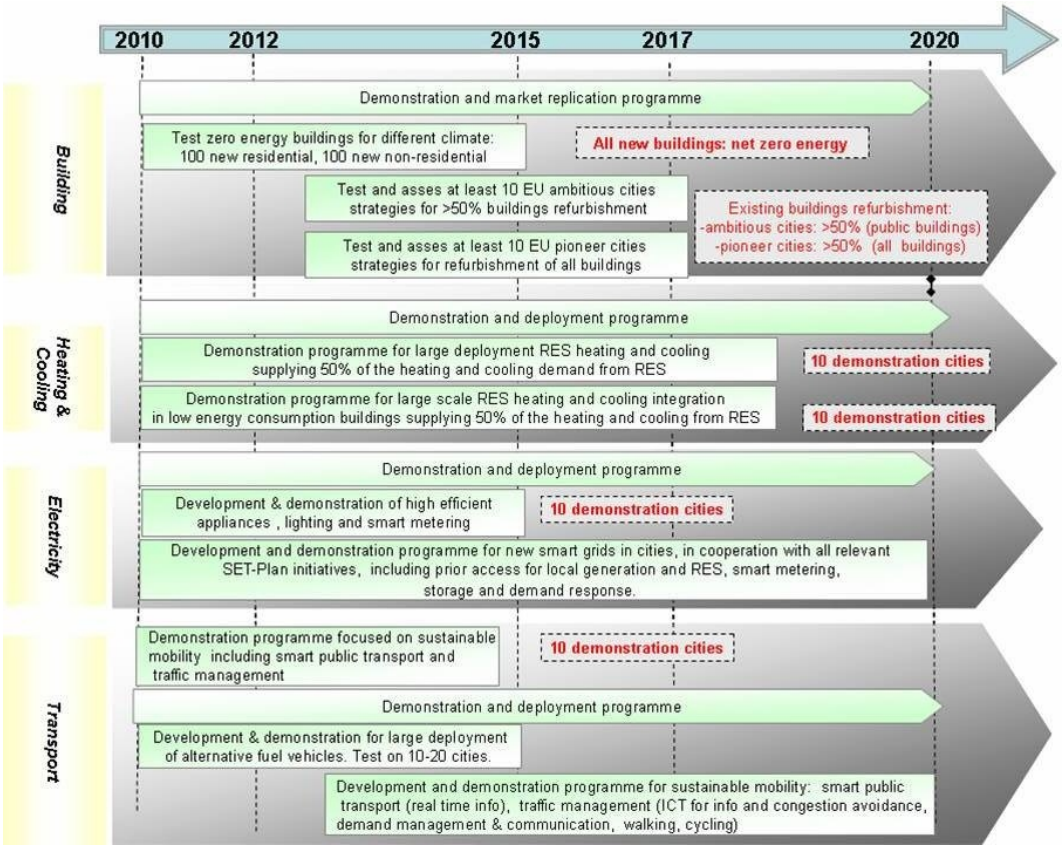

Figura 2. Hoja de ruta de la Ciudad Inteligente para Europa 2020 [5]

1.1.3. Internet de las Cosas

La primera referencia conocida al término «Internet de las Cosas» se debe a una presentación que Kevin Ashton realizó para Procter & Gamble en 1999 [7]. En aquél entonces parecía una buena idea asociar la reciente tecnología RFID (Identificación por Radio Frecuencia) con las posibilidades de su uso conjunto con la tecnología Internet.

En 2005 la ITU (*International Telecommunications Union*) rescató el término para su informe anual de política estratégica para Internet [8]. En este informe se recupera la visión del Dr. Mark Weiser acerca del futuro del desarrollo tecnológico de las TIC, y en particular del concepto que él mismo acuñó en 1988: la computación ubicua. Según este concepto el aumento de las capacidades de procesamiento debe ir acompañado de una reducción de su visibilidad. La idea de fondo es que

las tecnologías más arraigadas tienden a desaparecer, a confundirse con el tejido de las actividades cotidianas, llegando a hacerse indistinguibles de lo que resulta natural e intuitivo.

Cada vez son más los dispositivos interconectados, y los desarrollos actuales pretenden llevar este fenómeno un importante paso adelante. Así, mediante la integración de transceptores de corto alcance en una amplia variedad de cosas y en objetos cotidianos, habilitar nuevas formas de comunicación entre las personas y las cosas, y entre las cosas entre sí. Una nueva dimensión en el mundo de las tecnologías de la información y las comunicaciones (TIC), que añade a la propuesta de la conectividad para cualquiera en cualquier lugar y en cualquier momento, la nueva realidad de para cualquier cosa.

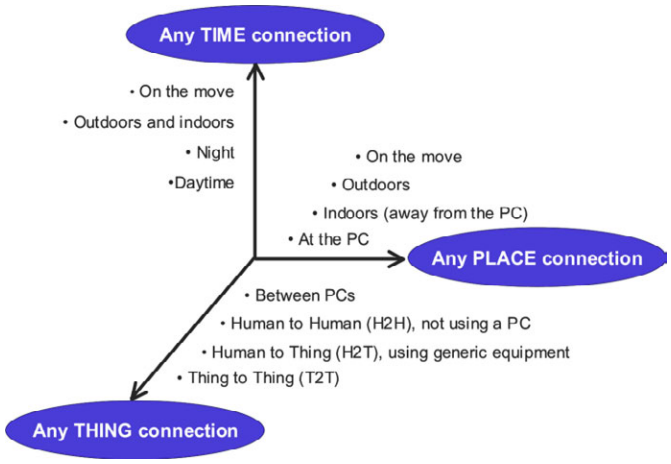

Figura 3. Una nueva dimensión (adaptado de Nomura Research Institue) [8]

Muchos son los dominios de aplicación donde se está observando el desarrollo de tecnologías que permiten la interconectividad entre cosas, objetos tradicionalmente aislados, que mediante el uso de las comunicaciones incrementan sus posibilidades. En la Figura 4 se describe el ecosistema de tecnologías, soluciones y paradigmas relacionados con Internet de las Cosas distribuidos por sectores industriales.

En conclusión, Internet de las cosas es una revolución tecnológica que representa el futuro de la computación y las comunicaciones, con un desarrollo multidisciplinar que implica importantes campos que van desde las redes de sensores hasta la nanotecnología.

Desde el uso de tecnologías que permitan la identificación de cualquier cosa de forma automática para su recolección y proceso digital, hasta la integración de mayores capacidades de procesado en dispositivos cada vez más reducidos. La capacidad de interoperación entre ellos será lo que les dé mayor utilidad, resultando en un beneficio mayor que la suma de las partes, en el que

Internet de las cosas será la tecnología que ofrezca la funcionalidad al resto para alcanzar la visión de un entorno de red completamente interactivo y sensible.

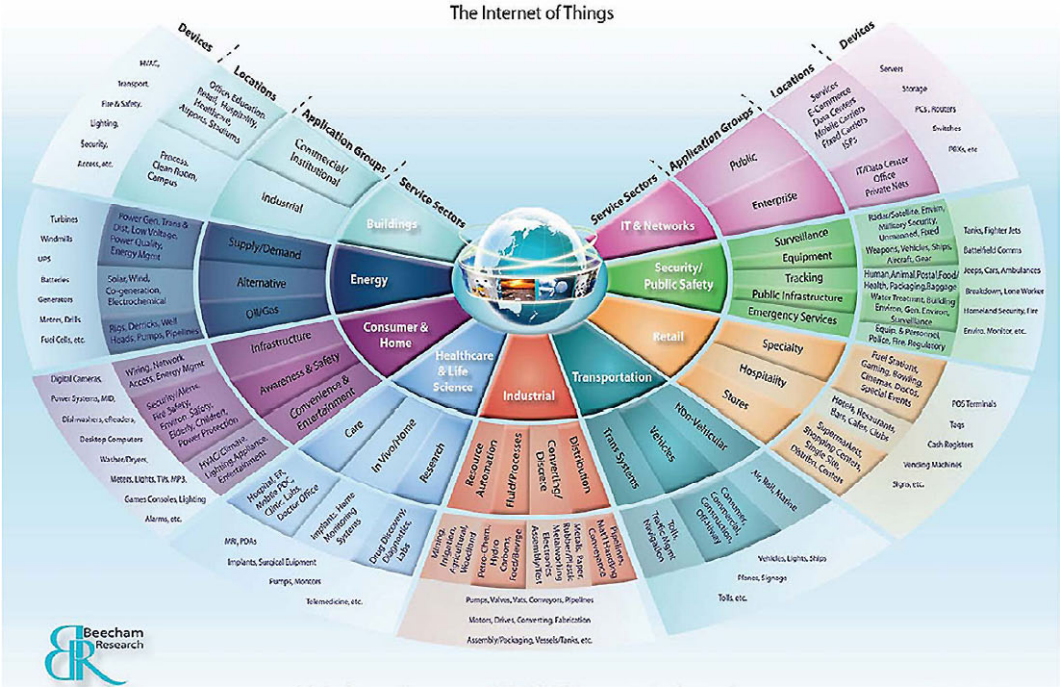

Figura 4. Ecosistema IoT en diferentes sectores industriales [9]

1.1.4. El proyecto Web of Objects

El proyecto Web of Objects (WoO) [10] aprobado dentro del marco del programa europeo ITEA2 [11] propone la simplificación del despliegue de objetos y aplicaciones, así como del mantenimiento y la operación de infraestructuras de edificación basadas en objetos inteligentes [12].

Desde la perspectiva de un control centralizado habitual en la domótica y la inmótica hasta la descentralizada ofrecida por Internet de las Cosas, el proyecto Web of Objects trata de definir una posición intermedia. Para ello propone encontrar el equilibro entre la apertura de los dispositivos que forman parte de la red y su protección, entre la accesibilidad pública a la información y el respeto a la privacidad.

Objetivos:

- Interoperatividad entre dispositivos y servicios a través de semántica.
- Adaptación de servicios basada en perfiles de usuario y contextos.

- Aumento de la seguridad a nivel de red, dispositivos y servicios.
- Detección y reconfiguración dinámica de servicios
- Cooperación de dispositivos en diferentes flujos de trabajo.

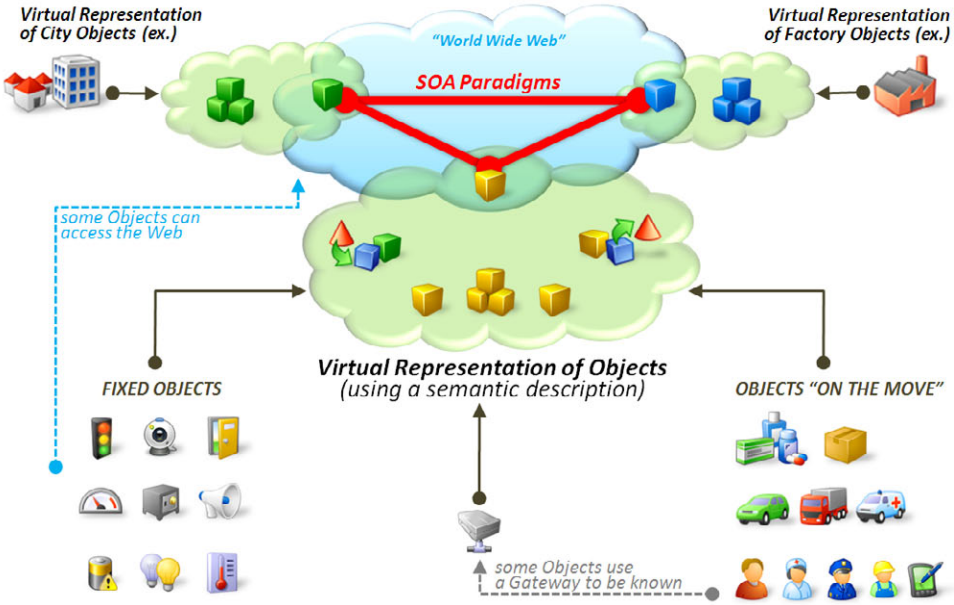

Figura 5. Enfoque del proyecto Web of Objects [13]

El objetivo general de WoO es simplificar el despliegue de objetos y aplicaciones, el mantenimiento y operación de las infraestructuras IoT dentro de los edificios, que por definición se encuentras definidas dentro de unas limitaciones estrictas en recursos como la capacidad de procesamiento, el ancho de banda y el suministro de energía.

1.1.5. Resumen

Web of Objects realiza una propuesta para que las cosas conectadas entre sí puedan interoperar mediante el uso de servicios web normalizados, facilitando su acceso tanto a personas como a otras cosas u objetos. Para lograr su objetivo se apoya en los avances sobre Internet de las Cosas, un paradigma tecnológico que ofrece la visión de un mundo de cosas interconectadas comunicándose entre sí y con las personas.

Estas tecnologías son de gran utilidad para alcanzar los objetivos de Europa 2020, y en particular para desarrollar las infraestructuras necesarias que permitan la construcción de ciudades inteligentes que cumplan con los objetivos necesarios para el desarrollo de la Ciudad del Futuro.

1.2. Objetivos

El Grupo de Redes y Servicios de Próxima Generación (GRyS) de la Universidad Politécnica de Madrid (UPM) ha estado desarrollando un middleware para redes de sensores y actuadores (WSAN) con su participación en anteriores proyectos de investigación.

Como participante en el proyecto WoO, la UPM plantea extender y actualizar dicho middleware para cumplir con la arquitectura y el framework WoO, de forma que sensores, actuadores, y sus servicios puedan integrarse en dicha arquitectura. En particular, siguiendo el enfoque de una Arquitectura Orientada a Servicios (SOA), se proyecta el uso de sensores virtuales a través de la orquestación de los servicios de sensores y actuadores simples; mejorar el rendimiento de la red para reducir su consumo; e implementar servicios de red y de seguridad para mejorar su capacidad de recuperación [14].

Este PFC, al estar enmarcado dentro de la participación de la UPM en el citado proyecto WoO, recoge parte de sus objetivos, y profundiza en ellos, quedando organizados como sigue:

- **Diseño y desarrollo de una pasarela de servicios IoT-A** basada en una solución SOA de tipo bus empresarial (ESB) empleando el middleware nSOM.
- **Diseño y puesta en servicio de un registro de servicios IoT-A** de acceso REST para consulta y actualización tanto automática por los servicios de la red como manual por parte del personal autorizado.
- **Estudio y desarrollo de un framework de orquestación** para el middleware nSOM. En proyectos anteriores ha quedado demostrado que el middleware nSOM es capaz de ofrecer servicios compuestos de tipo orquestado creados específicamente para escenarios predefinidos en los que el contexto era previamente conocido. En una evolución natural del middleware se propone y desarrolla una solución *framework* para la orquestación de servicios teniendo en cuenta la limitación de recursos de una red inalámbrica de sensores y actuadores (WSAN), y capaz de funcionar en escenarios donde el entorno pueda ser variable, adaptándose de forma dinámica según eventos.

1.3. Organización

El presente documento se encuentra organizado de la siguiente forma:

- **Capítulo 1:** El presente capítulo, en el que se ha abordado una breve introducción a los conceptos de Internet de las Cosas y de la Ciudad del Futuro, así como de los objetivos planteados dentro de un escenario concreto como el propuesto en el Proyecto Europeo de Investigación *Web of Objects* en el que se enmarca este PFC.
- **Capítulo 2:** Estudio detallado del marco tecnológico, o *estado del arte*, sobre el que se apoya la propuesta de desarrollo que se abordará en capítulos posteriores.
- **Capítulo 3:** Descripción detallada de la arquitectura *middleware* nSOM desarrollada en el Grupo de Redes y Servicios de Próxima Generación para la provisión de un mecanismo de intermediación en Redes Inalámbricas de Sensores, de aplicación para el escenario propuesto en el siguiente capítulo.
- **Capítulo 4:** Propuesta de un escenario, extraído del proyecto WoO, y de las provisiones tecnológicas para su abordaje, así como de las apropiadas métricas de evaluación de su validez dentro de unas especificaciones de diseño predefinidas.
- **Capítulo 5:** Conclusiones en base a los resultados expuestos en el capítulo previo, así como una breve exploración de posibles desarrollos futuros y sus aplicaciones.
- **Anexo I:** Documentación de las API para los componentes de la plataforma ESB.
- **Anexo II:** Documentación de las API de orquestación nSOM.
- **Anexo III:** Documentación de la API REST.
- **Anexo IV:** Comparativa de tecnologías de red.
- **Anexo V:** Comparativa de productos ESB.

1.4. Marco de desarrollo

Este Proyecto Fin de Carrera se enmarca dentro del Proyecto Europeo de Investigación *Web of Objects*, financiado por el subprograma de ayudas Avanza Competitividad I+D+I del Ministerio de Industria, Energía y Turismo (expediente TSI-020400-2011-29), y avalado por el programa de apoyo a la investigación ITEA2 (Information Technology for European Advancement) del organismo europeo EUREKA. [11].

Capítulo 2

Estado del arte en Internet de las Cosas y la Ciudad del Futuro

2.1. Conceptos tecnológicos

2.1.1. Redes inalámbricas de sensores y actuadores (WSAN)

Una red inalámbrica de sensores y actuadores (*Wireless Sensor and Actuator Network* o *WSAN*) [15] es una infraestructura compuesta por elementos capaces de *sentir* (detectar y medir) e intervenir en el entorno, procesar la información y comunicarse entre sí [16]. Estos elementos autónomos suelen recibir el nombre de nodos o motas.

Habitualmente en una WSAN existen cuatro tipos básicos de componentes:

a) Los nodos, es decir, los sensores y actuadores propiamente dichos.
b) La red de interconexión, que en el caso de una WSAN es de tipo inalámbrico.
c) Uno punto central donde se recolecta la información, conocido como sumidero.
d) Un conjunto de unidades de procesamiento para manejar los datos facilitados por los sensores, los eventos generados, el procesamiento de las consultas y toda serie de operaciones de utilidad como por ejemplo la minería de datos procedentes de la WSAN.

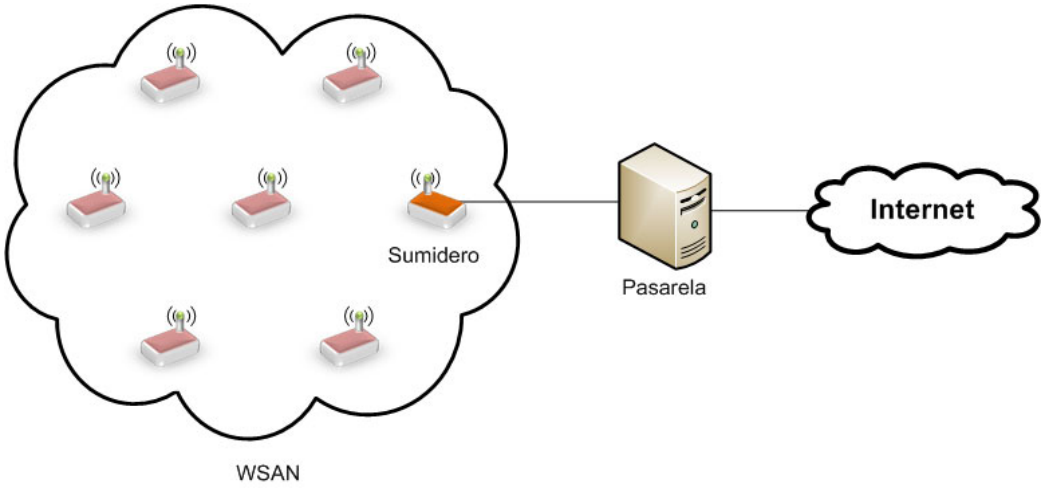

Figura 6. Arquitectura básica de una WSAN

2.1.1.1. Nodos de una WSAN

Un nodo es un dispositivo físico capaz de obtener información del entorno y/o interactuar con él en función de su rol como sensor y/o actuador. Para tal fin este tipo de dispositivos posee una arquitectura básica compuesta de cinco componentes principales mostrados en la Figura 7 [17].

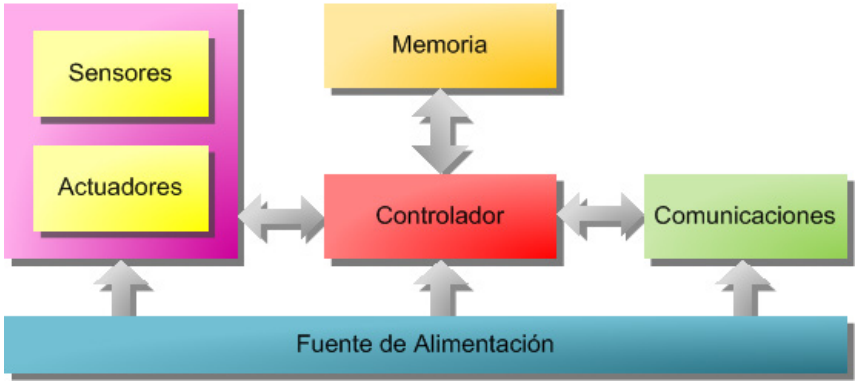

Figura 7. Visión general de los componentes de un nodo WSAN

Su descripción es la siguiente:

- **Controlador**: Un controlador para procesar la información relevante capaz de ejecutar código arbitrario.
- **Memoria**: Para almacenar el código a ejecutar y los datos de intercambio. Habitualmente se emplean diferentes tipos de memoria para separar datos y programa.
- **Sensores y actuadores**: La interfaz del nodo con el mundo físico, permitiéndoles observar o controlar los parámetros físicos del entorno.
- **Comunicación**: Para poder integrarse en una WSAN los nodos requieren de un componente que les permita enviar y recibir información sobre un canal inalámbrico.
- **Fuente de alimentación**: Por lo general para darle sentido práctico al uso de los nodos estos no suelen disponer de un suministro estable de energía, por lo que se requiere algún tipo de batería, generalmente recargable, e idealmente capaz de obtener energía del propio entorno donde se despliega el nodo.

2.1.1.2. Sumidero, pasarela y estación base

El sumidero es un tipo especial de nodo que recoge la información generada por el resto de nodos sensores de la WSAN. Este nodo puede formar parte de la propia red o no, y servir a su vez como pasarela a otras redes.

Habitualmente se usa un sumidero conectado a una unidad de procesamiento para dar acceso desde el mundo exterior a los sensores y actuadores que forman la red. Este conjunto de sumidero y unidad de procesamiento realiza las funciones de *gateway* o pasarela capturando la información que fluye desde y hacia la red, y conectándola a otras redes, especialmente a Internet.

2.1.2. Máquina a Máquina (M2M)

Las comunicaciones máquina a máquina (M2M por sus siglas en inglés) son las que se realizan entre dos o más entidades sin implicar necesariamente cualquier tipo de intervención humana directa. Por tanto los servicios M2M tratan de automatizar los procesos de decisión y comunicación [18].

Cualquier sistema M2M debe ser capaz de permitir la comunicación entre aplicaciones M2M que se encuentren en su dominio de red y de dispositivos M2M empleando múltiples esquemas de comunicación basados en un acceso IP.

De la misma forma cualquier objeto conectado debe ser capaz de comunicarse directamente con cualquier otro objeto mediante el uso de tecnologías P2P, incluso en la consideración de que cualquiera de los objetos receptores de una comunicación se encuentre o no disponible.

Además un sistema M2M debería abstraer la infraestructura de red del resto de partes del sistema, incluyendo los mecanismos de direccionamiento que se empleen.

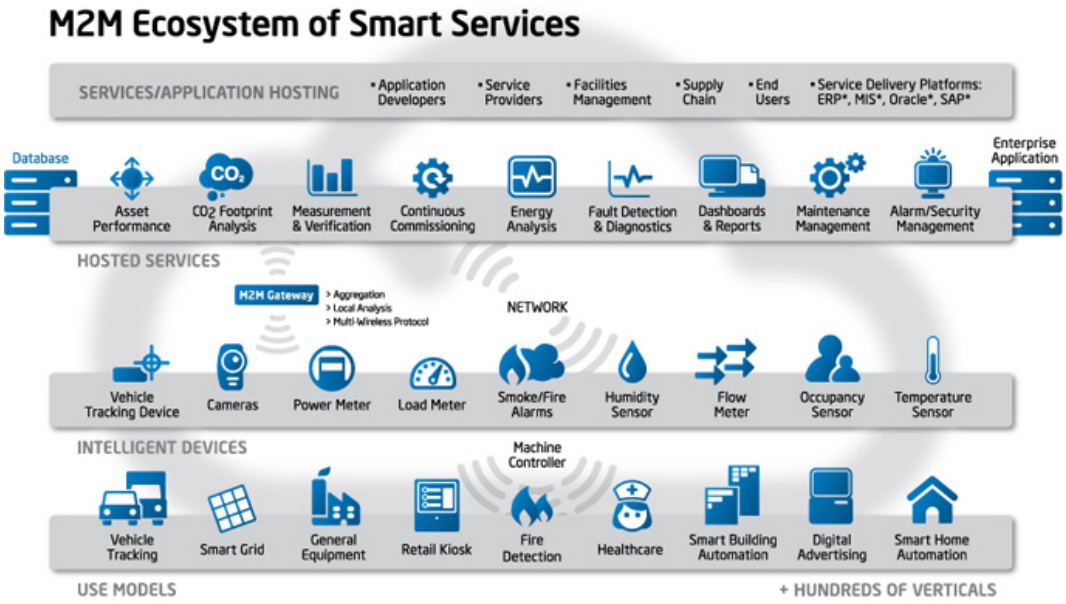

Figura 8. Ecosistema M2M [19]

2.1.2.1. Elementos clave de una red M2M

El instituto Europeo de Normas de Telecomunicaciones (ETSI por sus siglas en inglés) ha puesto en marcha una iniciativa de normalización de las tecnologías M2M [20]. Durante la celebración del *Mobile World Congress* de Barcelona en 2011, ETSI presentó su visión acerca de M2M, definiendo un conjunto básico de elementos clave [21]:

- **Dispositivo M2M**: Dispositivo capaz de responder a las peticiones de la información almacenada en ellos, o capaces de transmitir datos de forma autónoma.
- **Red de Área M2M (Domino de dispositivo)**: Provee la conectividad entre dispositivos M2M y pasarelas M2M.
- **Pasarela M2M**: Emplea las capacidades de un sistema M2M para garantizar la capacidad de interoperación de los dispositivos M2M y su interconexión con la red de comunicaciones.
- **Red de Comunicaciones M2M (Dominio de red)**: Las comunicaciones que ocurren entre las pasarelas M2M y las aplicaciones M2M. Por ejemplo: xDSL, LTE, WiMAX y WLAN.
- **Aplicaciones M2M**: Contienen la capa de intermediación donde la información pasa a través de varios servicios de aplicación y es utilizada por motores específicos de procesado de negocio.

2.1.2.2. Arquitectura de alto nivel

Dentro de su esfuerzo de normalización realizado por el ETSI se ha definido una arquitectura funcional cuya visión de alto nivel puede apreciarse en la Figura 9. Desde este punto de vista la arquitectura de alto nivel M2M incluye un Dominio de Dispositivos y Pasarelas, y otro Dominio de Red.

El dominio de Dispositivos y Pasarelas se compone de los siguientes elementos:

- **Dispositivo M2M**: Ya definido con anterioridad como uno de los elementos clave M2M, un dispositivo M2M puede conectarse al Dominio de Red de dos formas diferenciadas:
    - **Conectividad directa**: En este caso los dispositivos M2M se conectan directamente al Dominio de Red a través de la red de acceso. El dispositivo realiza directamente las funciones de registro, autenticación, autorización, gestión y aprovisionamiento con el Dominio de Red. Además este tipo de

dispositivos puede dar servicio a otros dispositivos que tenga conectados y que permanezcan ocultos para el Dominio de Red.

- **Conectividad a través de una pasarela como Intermediario de Red**: El dispositivo M2M se conecta al Dominio de Red a través de una pasarela M2M. Estos dispositivos se conectan con la pasarela a través de la Red de Área M2M. La pasarela a su vez actúa de intermediario desde el Dominio de Red hacia los dispositivos que tiene conectados.

- **Red de Área M2M**: facilita la conectividad entre los dispositivos y las pasarelas M2M. Estas redes pueden ser de múltiples tipos: IEEE 802.15.1 (Bluetooth), IEEE 802.15.4, ZigBee, ETF ROLL, ISA100.11a, PLC, M-BUS, KNX, X10, etc.
- **Pasarela M2M**: Una pasarela que ejecuta aplicaciones M2M empleando Capacidades de Servicio M2M. Actúa como intermediario entre los dispositivos M2M y el Dominio de Red, y además da servicio a otros dispositivos conectados que permanecen ocultos para el Dominio de Red.

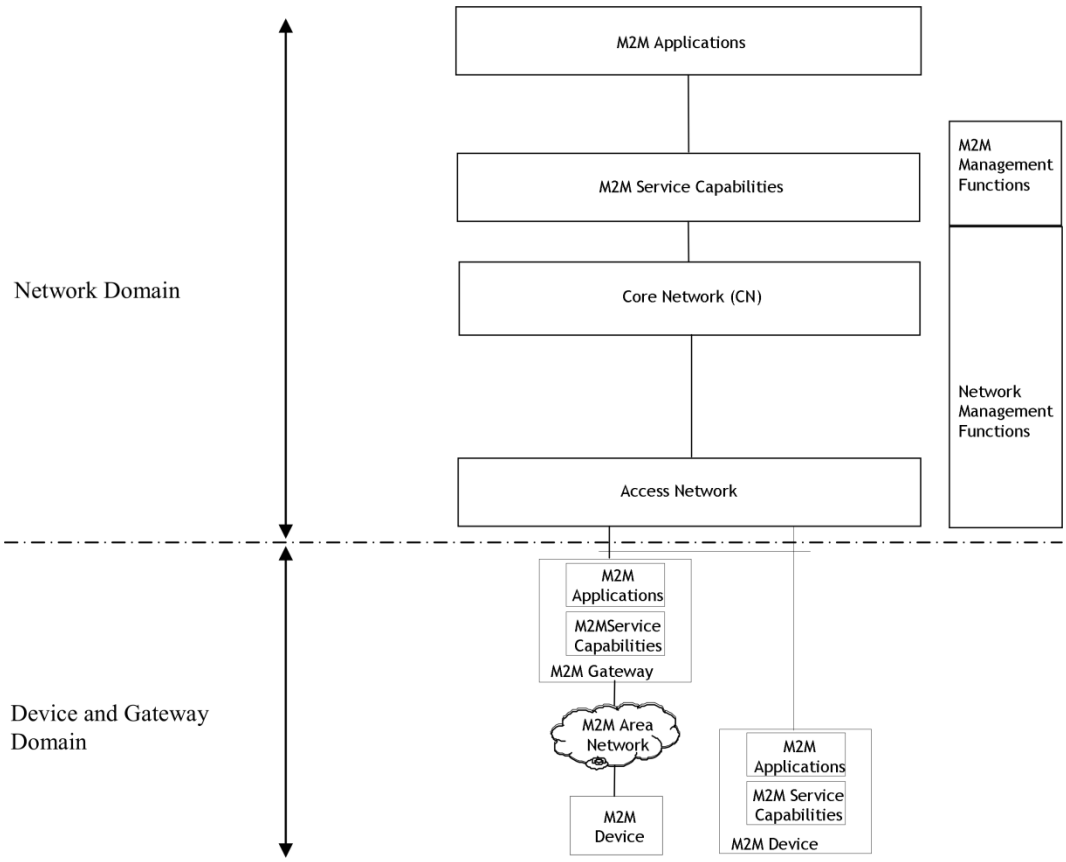

Figura 9. Arquitectura de alto nivel para M2M [22]

En cuanto a los elementos que componen el Dominio de Red son los siguientes:

- **Red de Acceso**: Red que permite al Dominio de Dispositivos y Pasarelas comunicarse con las redes centrales. Las redes de acceso pueden ser: xDSL, HFC, GERAN, UTRAN, eUTRAN, WLAN y WiMAX entre otras.
- **Redes centrales**: Proporcionan:
    - Conectividad IP básica y adicionalmente otras necesidades de conectividad.
    - Funciones de control de servicios y de red.
    - Interconexión con otras redes.
    - *Roaming*

  Las redes centrales incluyen, pero no están limitadas a: 3GPP CN, ETSI TISPAN CN.
- **Capacidades de Servicio M2M:**
    - Proporcionan funciones M2M compartidas entre diferentes Aplicaciones.
    - Exponen funciones mediante un conjunto de interfaces abiertas.
    - Usan las funcionalidades de las redes centrales.
    - Simplifican y optimizan el desarrollo de aplicaciones y su despliegue ocultando los aspectos específicos de la red.
- **Aplicaciones M2M**: Son aplicaciones que ejecutan la lógica del servicio y utilizan las Capacidades de Servicio M2M accesibles mediante interfaces abiertas.
- **Funciones de Gestión de Red**: Conjunto de todas las funciones necesarias para gestionar las redes de acceso y centrales, incluyendo el aprovisionamiento, la supervisión y la gestión de fallos entre otras.
- **Funciones de gestión M2M**: Conjunto de todas las funciones necesarias para gestionar las Capacidades de Servicio M2M en el Dominio de Red.

2.1.2.3. Marco de la arquitectura funcional

El marco de trabajo de la arquitectura funcional descrita por ETSI puede verse en la Figura 10. En el esquema descrito podemos observar como la interoperatividad entre un Dispositivo/Pasarela M2M y un nodo de la red M2M se lleva a cabo empleando una interfaz mId. Esta interfaz opera en un nivel de Capacidades de Servicio M2M (xSCL, siendo x «N» para el dominio de red, «D» para los dispositivos y «G» para las pasarelas).

A su vez el NSCL accede a las aplicaciones de red empleando una interfaz mIa, y puede conectarse a diferentes tipos de redes centrales para operar y comunicarse con otros elementos de un sistema M2M.

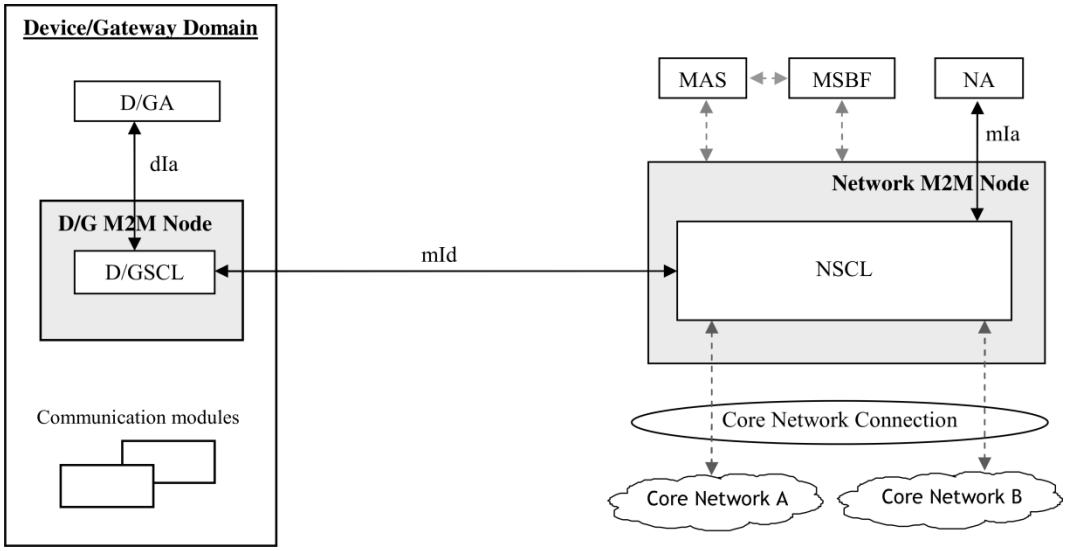

Figura 10. Marco de arquitectura funcional de las Capacidades de Servicio M2M [22]

Los diferentes componentes son:

- **Capacidades de Servicio M2M (M2M SC)**: Esta capa proporciona las funciones que van a exponerse en los puntos de referencia. Las M2M SC pueden conectarse a diferentes redes centrales y utilizar sus funcionalidades a través de sus interfaces públicas.

  Las capacidades de servicio pueden referirse a tres elementos de un servicio M2M: el Dominio de Red (NSCL), los Dispositivos M2M (DSCL) y las Pasarelas M2M (GSCL). En la Tabla 2 se recogen las capacidades de servicio provistas por la recomendación para cada uno de los elementos citados.

  Cada capacidad M2M corresponde a una recomendación para agrupar con lógica las funciones, pero no obliga a su implementación.
- **Aplicaciones M2M**: Pueden ser las Aplicaciones de Dispositivo (DA), de pasarela (GA) y de red (NA).
- **Punto de Referencia mIa**: Permite a las aplicaciones de red acceder a las M2M SC en el Dominio de Red. Su interfaz debe cumplir con la especificación de la ETSI [23].
- **Punto de Referencia dIa:** Permite a las aplicaciones de dispositivo ubicadas en un Dispositivo M2M acceder a las M2M SC que se encuentran en el mismo Dispositivo M2M o en una Pasarela M2M. Permite a las aplicaciones de pasarela acceder a las M2M SC de la misma pasarela. Su interfaz debe cumplir con la especificación de la ETSI [23].

- **Punto de Referencia mId:** Permite a las M2M SC de un Dispositivo o Pasarela M2M comunicarse con las M2M SC del Dominio de Red y viceversa. Su interfaz debe cumplir con [23].

Capacidades	Nivel de Capacidades de Servicio		
	Red	Pasarela	Dispositivo
Application Enablement (xAE)	X	X	X
Generic Communication (xGC)	X	X	X
Reachability, Addressing and Repository (xRAR)	X	X	X
Communication Selection (xCS)	X	X	X
Remote Entity Management (xREM)	X	X	X
SECurity (xSEC)	X	X	X
History and Data Retention (xHDR)	X	X	X
Transaction Management (xTM)	X	X	X
Compensation Broker (xCB)	X	X	X
Telco Operator Exposure (xTOE)	X	-	-
Interworking Proxy (xIP)	X	X	X

Tabla 2. Capacidades de Servicio M2M por elementos y niveles

2.1.3. Internet de las Cosas

Como ya se adelantaba en el Capítulo 1, la expresión «Internet de las Cosas» ha sido empleada por diferentes fuentes en la última década, englobando a una variedad de soluciones relacionadas de alguna forma con la comunicación entre objetos dotados de cierta inteligencia. El principal problema de estas soluciones es que responden a dominios verticales muy concretos, careciendo de interoperatividad entre ellos. Esto da lugar más a una variedad de «Intranets de las Cosas» que a lo que se pueda llegar a concebir como «Internet de las Cosas».

Dejando a un lado aplicaciones concretas, idealmente el concepto de «Internet de las Cosas» debe ser capaz de ofrecer, desde un punto de vista técnico, un marco común sobre el que definir cualquier solución necesaria en un dominio dado teniendo en cuenta la necesidad de que al menos en concepción debe ser capaz de interoperar con otras soluciones del mismo o diferentes dominios.

La Comisión Europea, a través de su 7º Programa Marco para la investigación y la innovación,

ha dado paso a varios proyectos con el fin de concretar y definir una generalidad sobre Internet de las Cosas, tanto en definiciones como en tecnologías. Destacan entre el resto los proyectos de la Iniciativa para Internet de las Cosas (IoT-I) [24], y el de Arquitectura para Internet de las Cosas (IoT-A) [25].

El proyecto IoT-I tiene como objetivo la creación de una comunidad IoT en Europa que mantenga y defina los conceptos de aplicación a las tecnologías IoT. Fruto de este proyecto es la IoT-Pedia [26], una base de datos de consulta pública en la que se recogen definiciones, proyectos, entidades y desarrollos IoT.

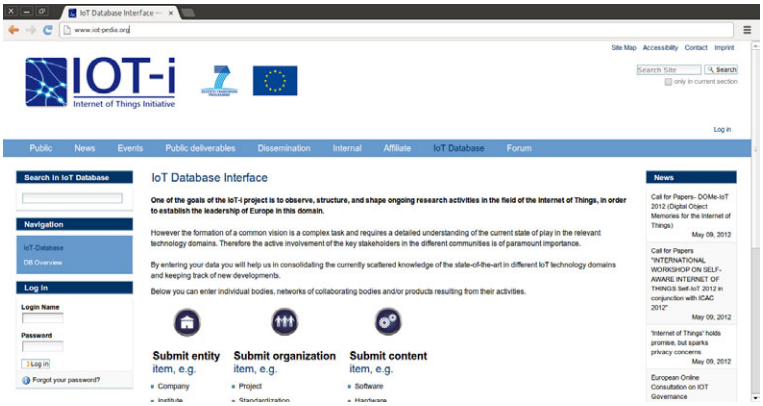

Figura 11. La IoT-Pedia

Por su parte el proyecto IoT-A se centra en el apartado tecnológico de diseño de IoT. Su trabajo se divide fundamentalmente en dos partes. En la primera se define un modelo de referencia que establece una comprensión común del dominio IoT, como se verá en la sección 2.1.3.1. La segunda parte del proyecto IoT-A es definir una arquitectura de referencia que permita servir de matriz a arquitecturas concretas.

Juntos, el modelo de referencia y la arquitectura de referencia forman el modelo de referencia de arquitectura IoT-A [27]. Su objetivo general se puede describir con una metáfora, como se ilustra en la Figura 12.

En la raíz se muestra una selección de protocolos y tecnologías de comunicación, así como otras tecnologías de dispositivos. Por otra parte en las hojas y flores del árbol IoT-A se encuentran las aplicaciones que se pueden construir utilizando estos elementos. Finalmente el tronco, la parte más importante, está formada por el modelo de arquitectura de referencia IoT-A. Esta parte es la fundamental para unir la raíz con las ramas, mostrando como el uso de las tecnologías de la base es empleado para construir las aplicaciones de la copa.

Figura 12. El árbol IoT-A [27]

2.1.3.1. Modelo de referencia

El modelo de referencia IoT-A proporciona los conceptos y definiciones necesarias sobre los que se construirán las arquitecturas IoT. Se compone de varios modelos inferiores que abarcan aspectos importantes del diseño IoT.

De todos los modelos que forman el modelo de referencia, el más importante es el llamado «Modelo de Dominio», pues en él se describen todos los conceptos relevantes para Internet de las Cosas. Todos los demás modelos se basan en estos conceptos.

Por encima de éste se sitúa el Modelo de Información, que define la estructura de la información manejada por un sistema IoT desde un punto de vista conceptual.

Finalmente se ubica el Modelo Funcional, que identifica grupos de funcionalidades centrados en conceptos definidos en el Modelo de Dominio. Estos grupos de funcionalidades a su vez responden a otros modelos, como es el caso del grupo funcional de comunicaciones y el Modelo de Comunicaciones, o el grupo funcional de seguridad y el Modelo de Seguridad.

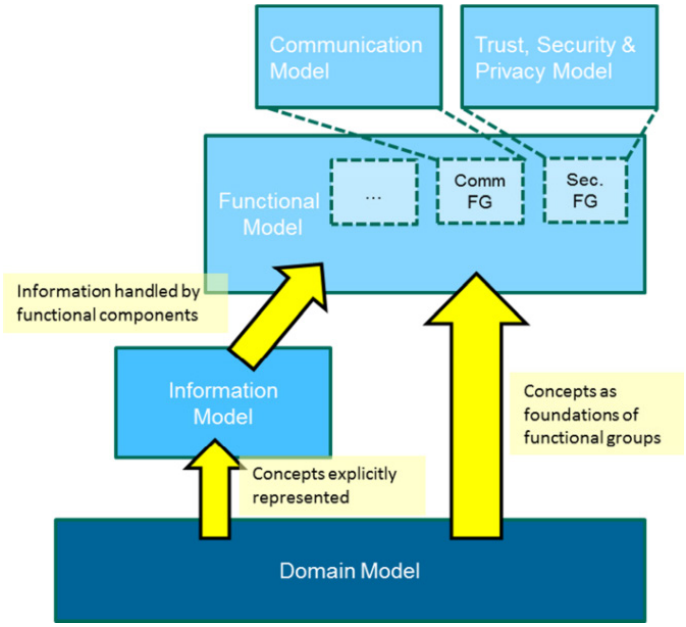

Figura 13. Interacción entre los sub-modelos de la referencia IoT-A [27]

2.1.3.1.1. Modelo de Dominio

Como se ha citado en la anterior sección, el modelo de dominio describe los conceptos pertenecientes al área de interés de IoT. Esto incluye los atributos básicos de los objetos de dominio, como por ejemplo su nombre e identificador, así como las relaciones entre ellos. Además se proporciona un léxico y una taxonomía comunes. Todo ello teniendo en cuenta la abstracción necesaria para garantizar su aplicación frente a futuros avances tecnológicos.

Este modelo se describe en el diagrama que se muestra en la Figura 14, utilizando el lenguaje de modelado UML.

Para describir el modelo se puede emplear un escenario genérico en el que un *Usuario* cualquiera requiere interactuar con una *Entidad Física* del mundo físico, posiblemente de forma remota. El *Usuario* puede ser una persona humana o cualquier tipo de *Artefacto Digital*.

La interacción con la *Entidad Física* se lleva a cabo invocando un *Servicio*. En el caso de un *Usuario Humano* el acceso al *Servicio* se realiza mediante algún tipo de cliente que no se muestra en el diagrama por claridad.

Para poder interactuar con ellas, las *Entidades Físicas* del mundo físico son representadas en el mundo digital mediante *Entidades Virtuales*. Este tipo de representaciones puede darse de múltiples formas: modelos tridimensionales, avatares, entradas en bases de datos, objetos de programación e incluso hasta cuentas activas en redes sociales.

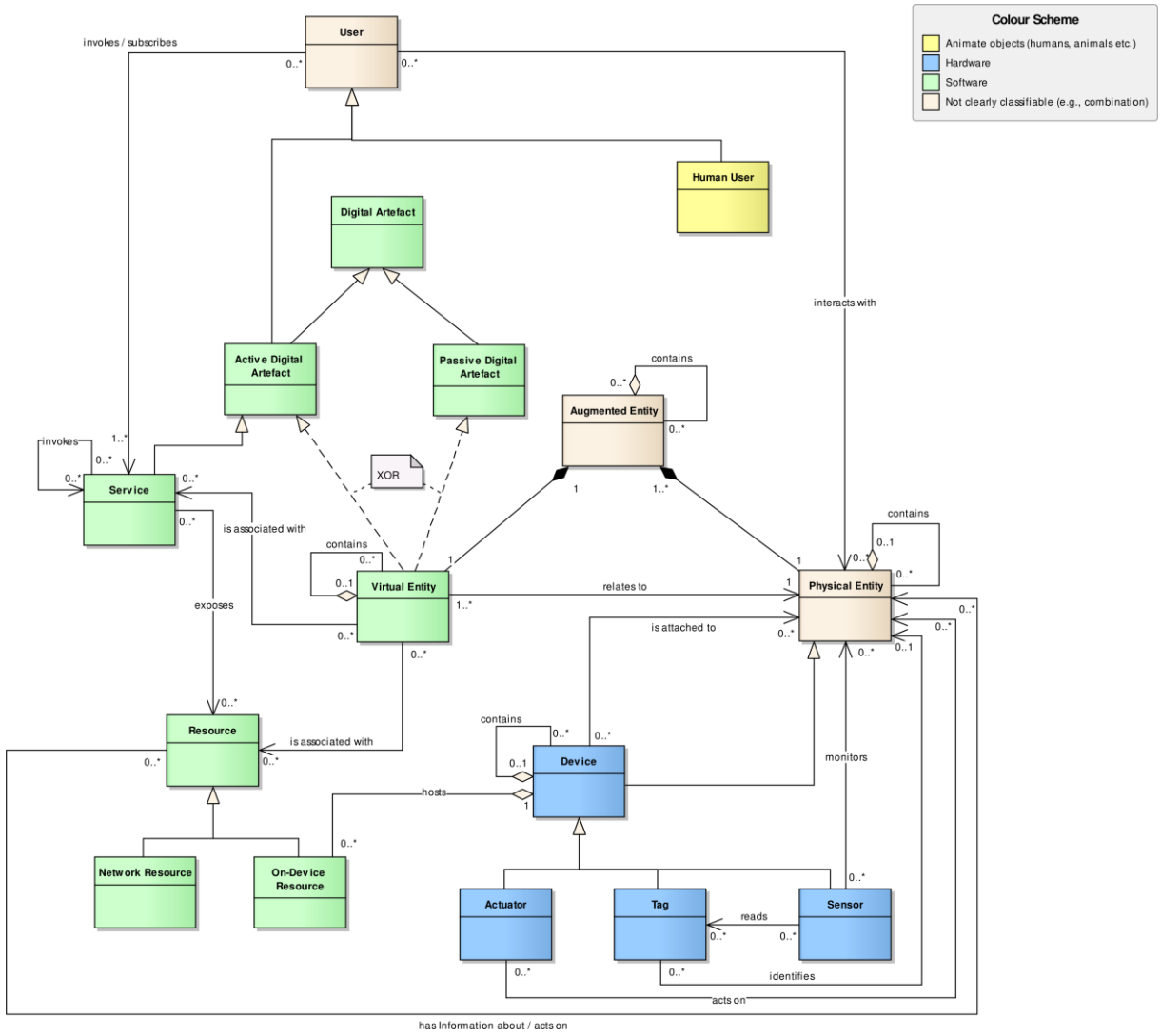

Figura 14. Modelo de Dominio IoT-A [27]

La multiplicidad de la relación de representación entre las *Entidades Virtuales* y las *Físicas* implica que generalmente una *Entidad Virtual* está asociada sólo a una *Entidad Física*, pero en la relación inversa, una *Entidad Física* puede estar representada por más de una *Entidad Virtual*.

En cualquier caso, desde el punto de vista del contexto IoT, las *Entidades Virtuales* deben tener al menos las dos propiedades fundamentales siguientes:

- Son *Artefactos Digitales* que pueden ser clasificados como activos o pasivos. Los *Artefactos Digitales Activos* ejecutan aplicaciones, agentes o *Servicios* que pueden acceder a otros *Servicios* o *Recursos*. Por su parte los *Artefactos Digitales Pasivos* son elementos pasivos de software, tales como las entradas en una base de datos, y otras representaciones digitales de la *Entidad Física* a la que representan.

- Idealmente las *Entidades Virtuales* son representaciones sincronizadas de un conjunto de aspectos o propiedades de las *Entidades Físicas*. Esto implica que los parámetros digitales relevantes que representan las características de la *Entidad Física* se actualizarán cuando ésta sufra cambios. Y lo mismo debería ocurrir en sentido contrario, manifestándose en la *Entidad Física* los cambios que afectasen a la *Entidad Virtual*.

La relación entre las *Entidades Físicas* y *Virtuales* se suele lograr mediante la integración, la conexión o simplemente la ubicación cercana a la *Entidad Física* de uno o más *Dispositivos* TIC que proporcionan la interfaz tecnológica necesaria para interactuar con ella y obtener información. De esta forma el *Dispositivo* aumenta las capacidades de la *Entidad Física* permitiéndole formar parte del mundo digital.

De nuevo, desde el punto de vista de IoT se conciben tres tipos de *Dispositivos*:

- *Sensores* que proporcionan información acerca de la *Entidad Física* que monitorizan. La información en este contexto va desde la identidad de la *Entidad Física* a las mediciones de su estado físico.
- *Etiquetas* que se emplean para identificar a las *Entidades Físicas* a las que están asociadas. El proceso de identificación se conoce como *lectura*, y se lleva a cabo por *Dispositivos Sensores* específicos, llamados *lectores*. Este proceso puede ser óptico, como es el caso de los códigos de barras y QR, o basado en radiofrecuencia, como los sistemas RFID.
- *Actuadores* que pueden modificar el estado físico de una *Entidad Física*.

Los *Dispositivos* pueden ser a su vez agregaciones de múltiples *Dispositivos* de diferentes tipos. Por ejemplo, puede existir un nodo sensor que contenga *Sensores* y *Actuadores*. Y por su funcionalidad deben ser capaces de operar tanto en el mundo físico como en el digital.

Desde el punto de vista del hardware, para operar en el dominio de IoT, los *Dispositivos* deben poseer ciertas capacidades de comunicación, procesamiento y almacenamiento, además de tenerse en cuenta la importancia de los recursos de energía disponibles con el objeto de considerar su autonomía operativa.

Las capacidades de comunicación están relacionadas con el tipo de datos intercambiados con el *Dispositivo* y la topología empleada en las comunicaciones. Estos aspectos son muy importantes en

el contexto de IoT, puesto que tienen un gran impacto en el consumo de energía, en la frecuencia de recolección de datos y en la cantidad de información que puede transmitirse. También debe tenerse en cuenta la seguridad, pues influye en las capacidades de comunicación al introducir una sobrecarga significativa.

En cuanto al procesamiento, su mayor impacto recae sobre la elección de la arquitectura, influyendo en las características de seguridad que puedan implementarse en los *Dispositivos*, y por supuesto en los recursos de energía de que dispongan.

Por su parte el almacenamiento hace referencia a la capacidad de contener la lógica a ejecutar en los *Dispositivos*, ya sea almacenada en el propio hardware, u obtenida de otros *Servicios*.

Recordemos que los *Dispositivos* son la forma en la que las *Entidades Físicas* obtienen su representación en el mundo digital a través de las *Entidades Virtuales*. Y estas a su vez tienen asociados *Recursos* y ofrecen *Servicios*.

Los *Recursos* son componentes de software que ofrecen algún tipo de funcionalidad, ya sea información o la posibilidad de cambiar aspectos del mundo físico o digital relativo a una *Entidad Física*.

Como se puede apreciar en la Figura 14, existen dos tipos de *Recursos*:

- *Recursos de Dispositivo* son aquellos que se encuentran en los *Dispositivos*, por lo general sensores y actuadores que permiten interactuar con el mundo físico. También pueden ser recursos de almacenamiento, por ejemplo para mantener un registro de las mediciones realizadas por los sensores.
- *Recursos de Red* son los que se ejecutan en un servidor dedicado o en la «nube», sin estar asociados a un hardware específico que les ofrezca una conexión directa con el mundo físico. Suelen ser recursos mejorados que ofrecen una mayor capacidad que la disponible en los *Dispositivos*, por ejemplo produciendo resultados agregados de la medición realizada por varios sensores, u ofreciendo una mayor capacidad de almacenamiento para el historial y servicio de caché.

Según OASIS los *Servicios* son el mecanismo para permitir el acceso a una o más capacidades, siendo este proporcionado por una interfaz predefinida y que es llevado a cabo en consistencia con las restricciones y políticas especificadas en la descripción del servicio [28]. En el Modelo de Dominio IoT los *Servicios* cumplen esta definición restringiéndose a servicios técnicos implementados por

software. En particular, los *Servicios* en IoT además de cumplir con interfaces bien definidas, ocultan la complejidad de acceso a la variedad heterogénea de *Recursos*.

Los *Servicios* IoT pueden ser clasificados según los siguientes niveles de abstracción:

- ***Servicios a nivel de Recurso*** son aquellos que exponen la funcionalidad de un *Dispositivo* accediendo directamente a sus *Recursos*.
- ***Servicios a nivel de Entidad Virtual*** son los que proporcionan acceso a la información en la *Entidad Virtual***.**
- ***Servicios Integrados*** son el resultado de la composición de servicios de nivel de *Recurso* y de nivel de *Entidad Virtual*, así como de cualquier combinación entre ellos.

2.1.3.1.2. Modelo de Información

El Modelo de Información define la estructura de toda la información que se maneja en un sistema a nivel conceptual. Esto incluye el modelado de los conceptos principales para el flujo de información, el almacenamiento y las relaciones entre ellos.

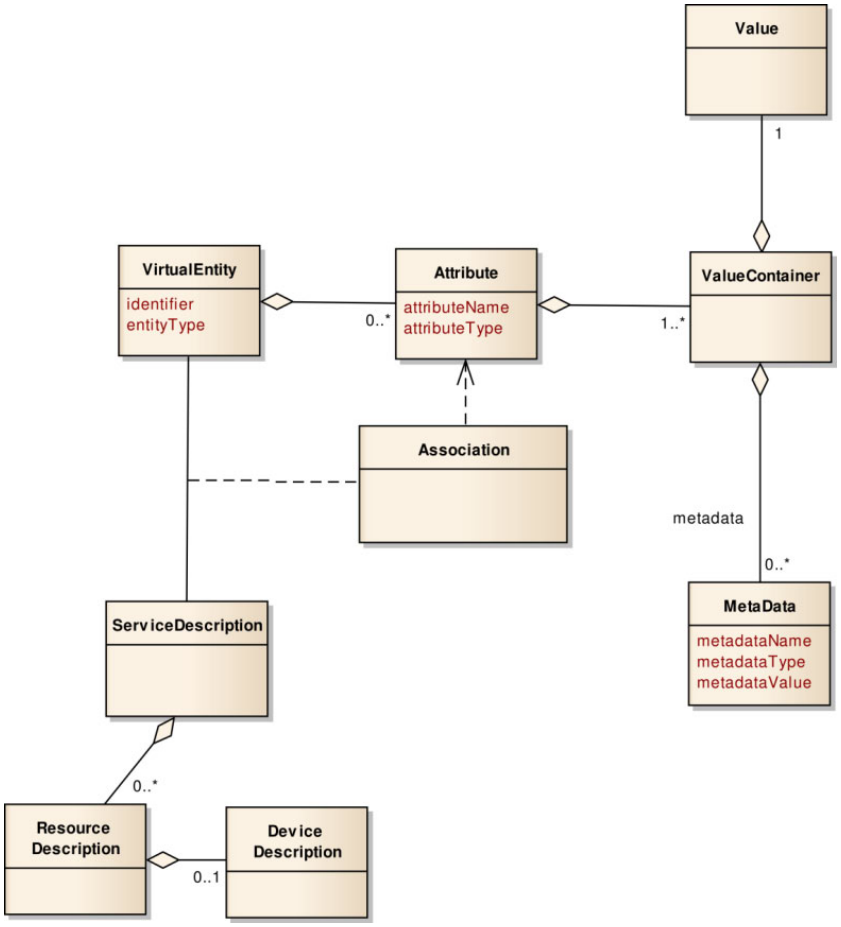

Figura 15. Modelo de Información IoT-A [27]

En particular, como puede observarse en la Figura 15, el Modelo de Información define los detalles de modelado de una Entidad Virtual, siendo esta la estructura de la información manejada y procesada por un sistema IoT.

Toda Entidad Virtual posee un identificador único o tipo de entidad, que define el tipo de representación de la Entidad Virtual, por ejemplo, un humano, un coche, o un sensor de temperatura. Además puede o no tener atributos, que su vez tienen un nombre, un tipo que lo define en un contexto semántico, y uno o más valores.

Además la Entidad Virtual se relaciona con una *Descripción de Servicio* para los servicios que proporciona. Esta descripción describe los aspectos relevantes del servicio, incluyendo su interfaz. Adicionalmente puede contener una o más *Descripciones de Recurso* para describir los recursos expuestos a través del servicio. Y estas a su vez pueden contener información sobre el dispositivo en el que se encuentran.

La relación entre el Modelo de Dominio y el de Información se muestra en la Figura 16.

Existen otros modelos de información definidos por el proyecto IoT-A que no han sido publicados o están pendientes de desarrollo en el momento de escribir esta memoria:

- **Modelo de Entidad**: Especifica los atributos y características de los objetos del mundo real que son representados por su contraparte virtual. Por cada atributo especificado en el modelo, se pueden asociar servicios que sean capaces de proporcionar y manipular la información acerca de él.
- **Modelo de Recurso**: Contiene la información que es esencial para identificar unívocamente a los recursos y clasificarlos por tipo. La especificación detallada de la descripción de los *Recursos* se describe en una especificación aparte dentro del proyecto IoT-A [29].
- **Modelo de Descripción de Servicios**: El Modelo de Servicios se define en [29], determinándose que la descripción de los servicios se realizará empleando el Lenguaje Unificado de Descripción de Servicios (USDL) siguiendo una estructura común.
- **Modelo de Eventos**: En el momento de escribir esta memoria aún no se ha especificado la representación y procesamiento de eventos, esperándose publicar en el futuro en el Entregable 2.6 del proyecto IoT-A.

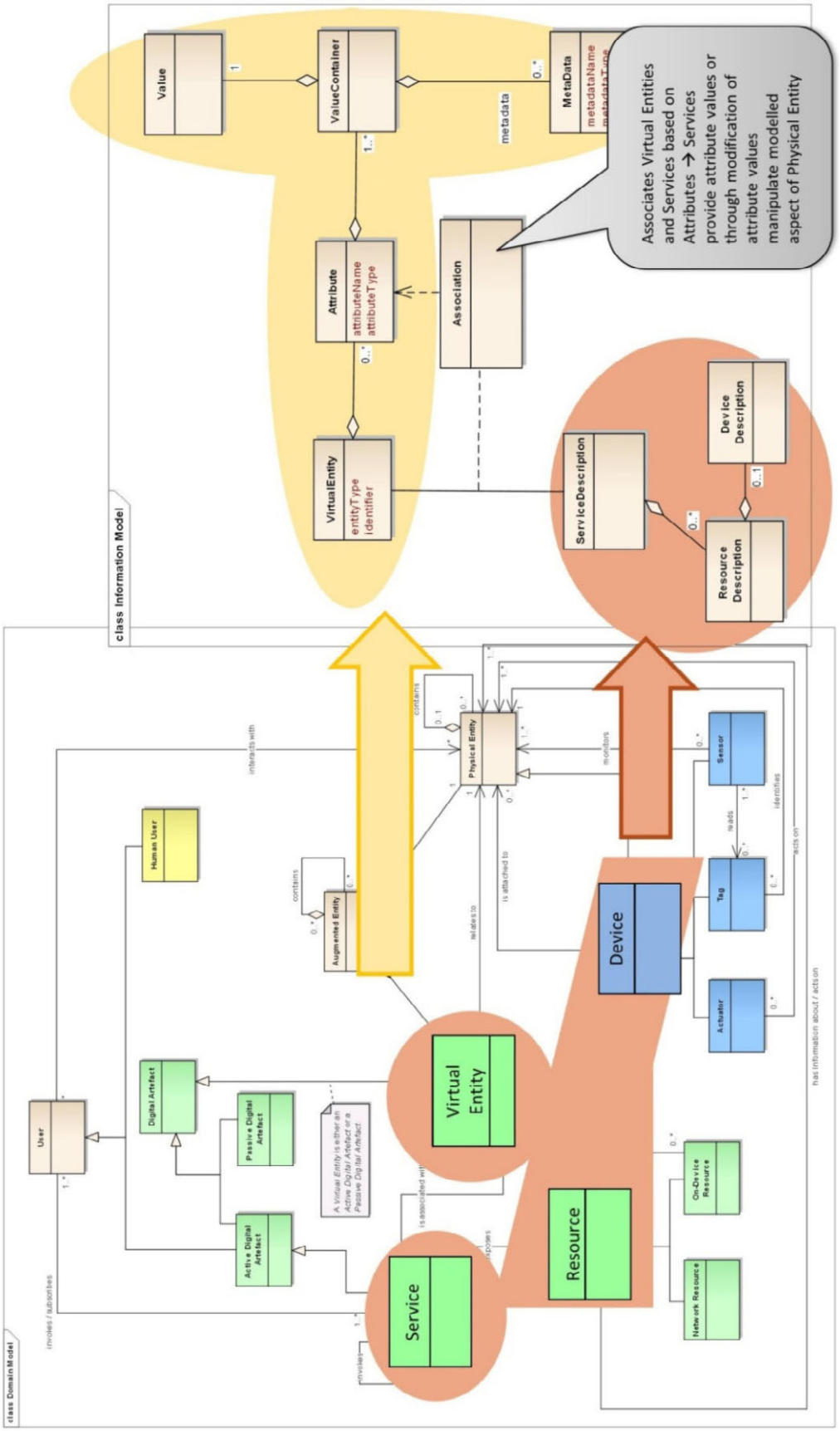

Figura 16. Relación entre los modelos de Dominio e Información IoT-A [27]

2.1.3.1.3. Modelo Funcional

La definición de modelo funcional empleada en el proyecto IoT-A deriva de la referencia descrita en el modelo de referencia para arquitecturas orientadas a servicios de OASIS [28], y que en el modelo de referencia de arquitectura para IoT [27] se describe como el marco abstracto para la comprensión de los principales grupos de funcionalidad de un entorno IoT-A y sus relaciones. Bajo este marco se define la semántica común.

El diagrama del modelo funcional se representa en la Figura 17, donde se detallan los grupos de funcionalidad de consideración para el modelo IoT-A.

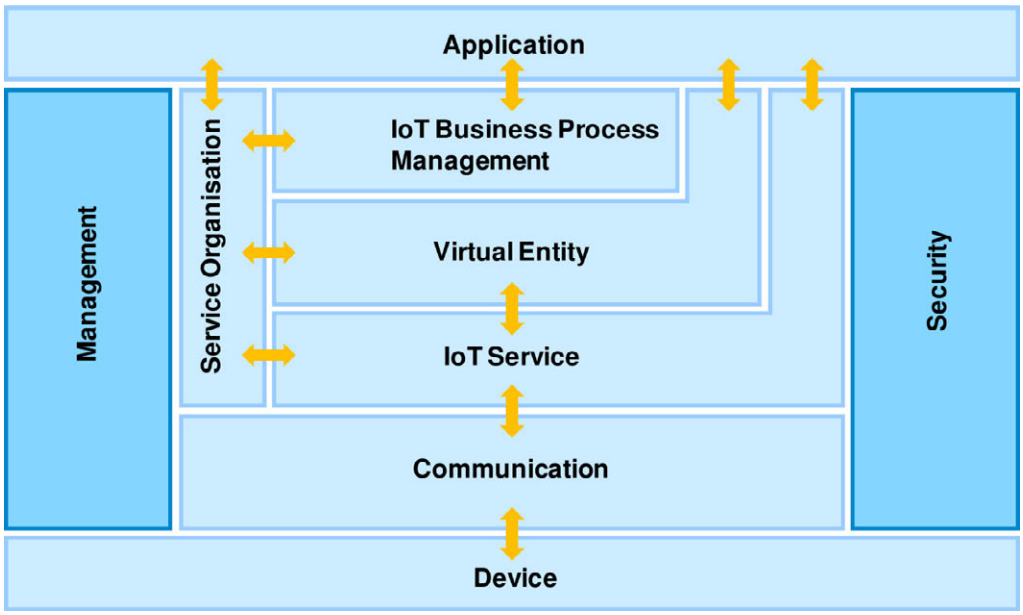

Figura 17. Modelo Funcional IoT-A [27]

Como se puede apreciar en la representación del modelo, existen siete capas longitudinales de grupos de funcionalidad, así como otras dos transversales. Estas últimas proporcionan funcionalidades que se requieren por cada uno de los grupos funcionales con los que coinciden a nivel. Por simplificación de la representación no se han incluido las relaciones entre estos grupos transversales y el resto, como sí se pueden apreciar mediante flechas los flujos de comunicación existentes entre el resto de los grupos.

Los grupos de funcionalidad son, en detalle, los siguientes:

- **Gestión de Procesos de Negocio IoT (IoT Business Process Management)**: Hace referencia a la integración de los sistemas tradicionales de gestión de procesos de negocio.

- **Organización de Servicios (Service Organization)**: Es el grupo funcional central que opera como un concentrador de las comunicaciones entre otros grupos funcionales. Se emplea para la composición y orquestación de servicios en diferentes niveles de abstracción. En definitiva actúa como un intermediario que puede enlazar las peticiones de servicio de alto nivel, incluso de aplicación, con los servicios básicos de los recursos IoT.
- **Entidad Virtual (Virtual Entity)**: Contiene las funciones necesarias para interactuar con el sistema IoT mediante Entidades Virtuales, así como las funcionalidades para el descubrimiento y la búsqueda de servicios que puedan ofrecer información sobre ellas, o que permitan su interacción.
- **Servicio IoT (IoT Service):** Contiene los servicios IoT y sus funcionalidades para el descubrimiento, búsqueda y resolución de nombre de los servicios IoT.
- **Comunicación (Communication):** Engloba todas las necesidades de comunicación de un sistema IoT-A.
- **Gestión (Management):** Combina todas las funcionalidades que son necesarias para mantener un sistema IoT. A tal fin se definen al menos cuatro objetivos básicos de alto nivel:
    - **Reducción de costes.**
    - **Atención de situaciones de uso imprevistas.**
    - **Gestión de errores.**
    - **Flexibilidad.**
- **Seguridad (Security)**: Es el responsable de garantizar la seguridad y la privacidad de los sistemas acordes a IoT-A.

2.1.3.1.4. Modelo de Comunicación

El modelo de comunicación se centra en la definición de los paradigmas de comunicación necesarios para conectar las entidades descritas en el Modelo de Dominio. Se propone un esquema similar al modelo de referencia básico OSI [30]. Además se describen los esquemas de comunicación que pueden ser aplicados a los diferentes tipos de redes en IoT.

2.1.3.1.4.1. Pila de Comunicaciones

La pila de comunicaciones que se propone, además de tomar de referencia la de OSI, se inspira en la pila de comunicaciones Internet. En la Figura 18 se muestran las diferentes pilas de comunicaciones de referencia, así como la propuesta para un sistema IoT-A.

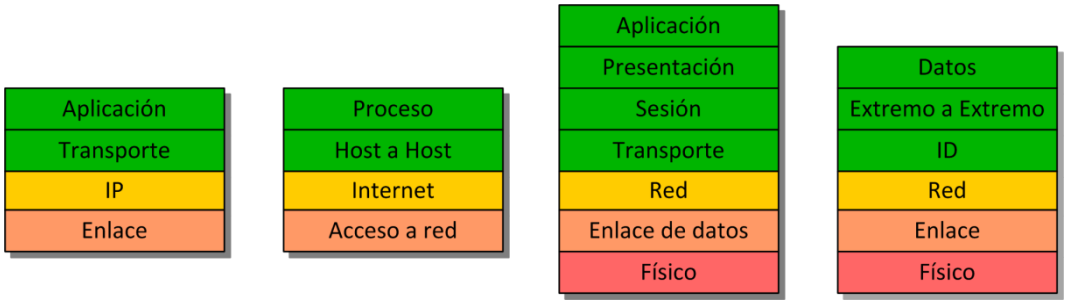

Figura 18. Comparativa de pilas de comunicaciones [27]

Las dos primeras pilas corresponden a diferentes visiones del conocido como modelo Internet. La primera es precisamente la pila Internet tal y como la conocemos hoy en día. A su lado se encuentra la pila tal y como se definió originalmente en el Departamento de Defensa para el proyecto ARPA y su evolución.

Le sigue la pila de siete niveles OSI, y en último lugar la propuesta IoT-A. En esta propuesta se pueden distinguir los siguientes niveles:

- **Físico**: Se emplea la misma definición que en el modelo OSI, es decir, la ruta de comunicación en el medio físico entre dos o más entidades físicas junto a las facilidades necesarias en el nivel físico para la transmisión de datos. La convergencia de soluciones heterogéneas de ser manejada por los niveles superiores.
- **Enlace**: Con el objetivo de manejar la heterogeneidad de las tecnologías de red que se aplican en el campo de la IoT, esta capa debe permitir la diversidad de protocolos. Pero a la vez debe proporcionar a los niveles superiores capacidades e interfaces uniformes.
- **Red**: De nuevo esta capa ofrece las mismas funcionalidades que la homónima del modelo OSI, con la extensión de requerir que cualquier posible solución de red obedezca a un paradigma común de comunicaciones. Idealmente en la propuesta de protocolo IoT se sugiere en este nivel el uso del protocolo IP, si bien existen otras soluciones tecnológicas en el mercado que pueden cumplir los objetivos en escenarios específicos.
- **ID**: O más concretamente el *Identificador de Entidad Virtual*, es el primer punto de convergencia en la pila de comunicaciones propuesta. Esta capa proporciona un marco de resolución común para IoT, y los servicios de seguridad, autenticación y alto nivel la utilizarán para obtener un direccionamiento común de los diferentes dispositivos y tecnologías de una red IoT.

- **Extremo a extremo**: Esta capa se encarga de la transferencia segura, el transporte y las funcionalidades de traducción, el soporte de los proxies/pasarelas y el ajuste de los parámetros de configuración cuando se usan diferentes entornos de red. Al ubicarla por encima de las capas de ID y de red se logra un bloque final de construcción acorde al modelo de comunicaciones M2M.
- **Información:** En la cima de la pila de comunicaciones propuesta se define la capa de datos o de información como el punto de entrada a la red. El propósito de esta capa es asegurarse que toda información generada por cualquier objeto de un sistema IoT es tratado de acuerdo al Modelo de Información.

Una característica destacable de la pila propuesta es que las pasarelas pueden implementarse hasta el nivel Extremo-a-Extremo con el objetivo de facilitar la comunicación de los dispositivos con características limitadas:

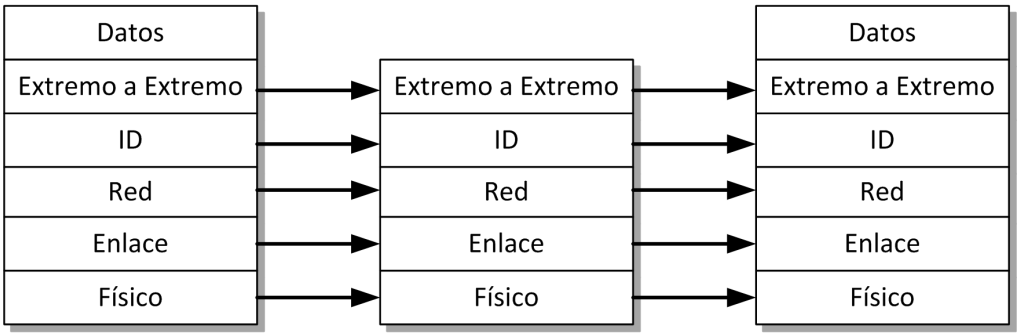

Figura 19. Pasarela IoT-A [27]

2.1.3.1.4.2. Modelo de canal para las comunicaciones IoT

El siguiente paso en el modelo de comunicaciones es definir y componer un modelo de canal basado en el modelo de comunicaciones de Shannon-Weaver.

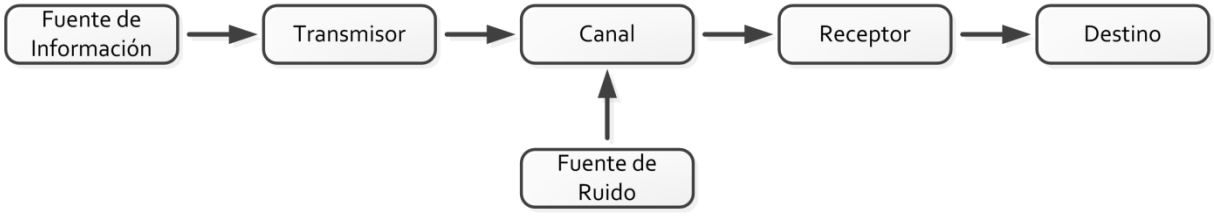

Figura 20. Diagrama del modelo general de comunicaciones Shannon-Weaver.

Considerando el contexto definido en el Modelo de Dominio IoT-A, se pueden agrupar los componentes del modelo de canal de forma que la fuente de información y el transmisor corresponden a la *Entidad Digital*, así como el receptor y el destino son el equivalente al usuario, ya sea este un servicio, un humano u otra entidad digital remota.

Queda por tanto por definir el canal dentro del contexto IoT, donde puede adquirir múltiples formas. Para tal fin se emplea una abstracción basada en el teorema de Shannon-Hartley, que por definición se debe aplicar independientemente a cada enlace que forme parte de la ruta entre el emisor y el receptor.

$$C_l = B_l \log\left(1 + \frac{S_l}{N_l}\right)$$

Esta fórmula nos da la capacidad del canal C en función de su ancho de banda B y su relación señal-ruido S/N. Y dado que esta métrica resulta en una función cóncava el cálculo de la capacidad resultante de la agregación de varios enlaces puede calcularse aplicando la siguiente regla:

$$C_{i,k} = min(C_{i,j}, C_{j,k})$$

Es decir, que dados dos canales adyacentes conectados mediante el uso de una pasarela, la capacidad total del enlace resultante es el mínimo valor de las capacidades individuales de cada segmento del enlace. Este cálculo resulta de extrema utilidad cuando se debe dimensionar la pasarela que comunica sendos canales. El resultado se limita a los enlaces que son conocidos para el sistema. Por ejemplo, cuando una red IoT se conecta a Internet, es imposible conocer con antelación cuál será la capacidad del canal correspondiente a dicho enlace. En tales casos es correcto asumir que ésta nunca será mayor que la de los enlaces conocidos, evitando así sobredimensionar la pasarela de forma innecesaria.

Esta es una importante cuestión a tener en cuenta con respecto a la planificación de las comunicaciones en un sistema IoT. Habitualmente estamos acostumbrados a concebir el uso de pasarelas para conectarse a Internet en la forma que se describe en la Figura 21:

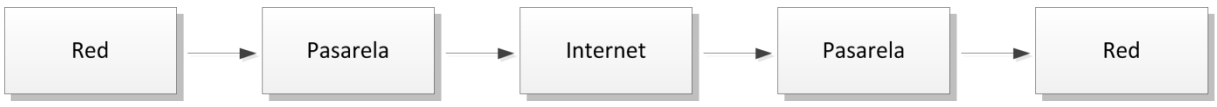

Figura 21. Modelo clásico de interconexión de redes en Internet

Pero en un sistema IoT las posibles redes se dividen en dos tipos que deben ser tenidos en cuenta, tal y como se recogen en la Tabla 3.

Tipo de red	Características	Ejemplos
Sin restricciones o limitaciones (NTU)	Enlaces de comunicación de alta velocidad, en el rango de transferencias con una tasa igual o superior al Mbit/s.	Internet
	Bajas latencias en el nivel de enlace, más afectadas por las congestiones en la red que por las propias características físicas de los elementos de la misma.	
Restringidas o limitadas (NTC)	Enlaces de comunicación de baja velocidad, en el rango de transferencias inferior al Mbit/s	LoWPAN
	Grandes latencias, debidas entre otros a una tecnología de nivel físico de baja capacidad y a las políticas de ahorro de energía de los dispositivos que operan en estas redes.	

Tabla 3. Tipos de redes en IoT según características del canal

Las redes heterogéneas son en realidad una combinación de redes con y sin limitaciones enlazadas mediante el uso de pasarelas, aunque éstas sean referidas con otros nombres.

Con estas consideraciones el modelo más simple de una red IoT es el de una red limitada, por ejemplo una WSAN, siendo entonces que el canal en el modelo de comunicaciones IoT sería la propia WSAN, esto es, la red limitada.

El canal dentro de este modelo depende por tanto de las redes que intervengan en la comunicación extremo a extremo. Por ejemplo, cuando dos redes limitadas se interconectan entre sí mediante una pasarela, el canal es la «nube» formada por ambas redes y la propia pasarela. Un ejemplo de este tipo de distribución puede consultarse en el trabajo de Alexandra Cuerva [31], en el que trata la interconexión de una WSAN y una red Bluetooth en el dominio de IoT.

Extrapolando la idea al modelo de interconexión en redes Internet, el canal IoT para la comunicación entre dos NTC empleando Internet como intermediario se ilustra en la Figura 22.

Figura 22. Canal IoT entre redes NTC empleando Internet como intermediaria

2.1.3.1.4.3. Relación entre el Modelo de Comunicación y el de Información

La relación entre los modelos de comunicación y de información se muestra en la Figura 23, empleando diferentes colores para representar las características obligadas del modelo de información que deben ser tenidas en cuenta en función del tipo de red.

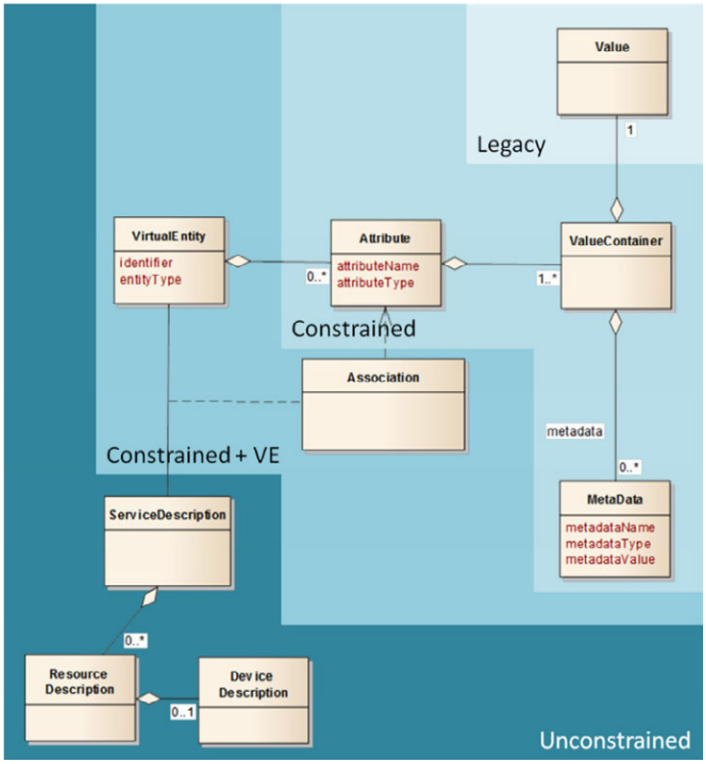

Figura 23. Relación entre los modelos de comunicación y de información en IoT [27]

Es importante tener presente que cualquier dispositivo integrado en un sistema IoT-A debe ser descrito con todas las características del modelo de información. Así pues, cuando éste se encuentre en una red limitada que no permita la implementación directa del modelo por completo, la pasarela que ofrezca la conexión de la red al sistema deberá complementar las partes que falten.

2.1.3.1.5. Modelo de Confianza, Seguridad y Privacidad

La confianza, la seguridad y la privacidad son cualidades horizontales en un sistema IoT, y describen la interacción entre dos entidades del Modelo de Dominio, el *Servicio* y el *Usuario*, así como su relación con los componentes de la infraestructura de Seguridad y Resolución [32].

2.1.3.1.5.1. Confianza

La confianza es una cualidad esencial de los sistemas IoT. La definición que se maneja dentro del modelo de referencia está basada en el trabajo de Diego Gambetta «*Trust: Making and Breaking*

Cooperative Relations» [33]. En particular se hace referencia al nivel de probabilidad subjetivo evaluado por el usuario con el que un sistema IoT realizará una acción particular o manifestará un comportamiento dado en un contexto que le es de relevancia. En otras palabras, la confianza está relacionada con lo que el usuario espera del sistema.

En lo que respecta a un sistema IoT, existen al menos dos niveles de confianza: el nivel de red y el de aplicación. El nivel de red está relacionado con la integridad de los procesos de comunicación, y en particular de encaminamiento, como por ejemplo que los mensajes sean entregados de la forma esperada y en un tiempo aceptable.

En cuanto al nivel de aplicación, se consideran importantes las cualidades de la información, de autenticación de extremo y no repudio, de confidencialidad y de políticas de privacidad y acceso a la información. La consideración de estas cualidades requiere de la definición de los adecuados modelos de confianza.

El diseño de los modelos de confianza debe iniciarse con el análisis de requerimientos, riesgos y contexto, detallando con claridad la forma en la que se entiende la confianza en el sistema, cómo debe ser medida, y cómo se deben manejar las relaciones entre las entidades participantes según su evaluación de confianza.

Existen una serie de aspectos de obligado cumplimiento en la definición de los modelos de confianza de un sistema IoT:

- El dominio del modelo de confianza debe definir de forma específica el conjunto de sujetos a los que es de aplicación el modelo. Esta definición puede estar basada en la suscripción, o por el contexto físico o de la red.
- Se deben definir los mecanismos de evaluación de la confianza con el objetivo de disponer de un método válido para el cálculo del grado de confianza de un sujeto.
- Las políticas de comportamiento deben definir como los sujetos que usen el modelo pueden interactuar con otros sujetos que puedan ser evaluados siguiendo el mismo modelo.
- Existe una figura de confianza que es confiable por defecto por todos los sujetos del modelo, y que es empleada para evaluar la confianza de terceras partes.
- Considerando la figura de confianza, debe existir una federación de confianza, de manera que se permita la interoperabilidad entre sujetos que se encuentren bajo diferentes modelos de confianza.

- Es esencial que exista soporte a las tecnologías M2M, donde la interacción autónoma entre máquinas es habitual. Estas máquinas necesitarán identificar y acceder a recursos de forma dinámica, por lo que se requieren pasos específicos para que puedan evaluar de forma autónoma el grado de confianza de otras máquinas.

2.1.3.1.5.2. Seguridad

Las características y niveles de seguridad que se describen en el modelo de referencia se muestran en la Figura 26. La descripción en detalle de las características se da al margen de la arquitectura de referencia, en [32], salvo ciertas cuestiones de relevancia que afectan al grupo correspondiente a la seguridad en las comunicaciones.

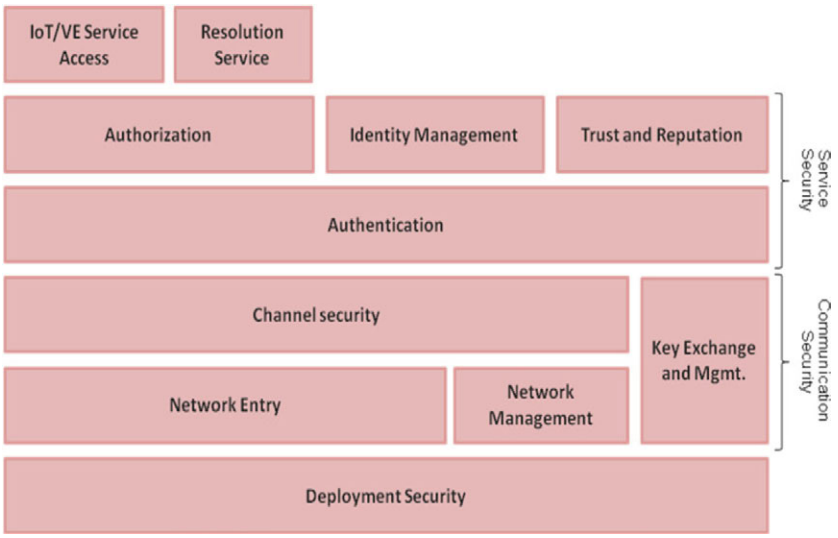

Figura 24. Características de seguridad y niveles en IoT-A [27]

Precisamente las comunicaciones en un sistema IoT implican una mayor dificultad para establecer medidas de seguridad a nivel de protocolo. El problema reside en que un sistema IoT, como ya se ha descrito en el modelo de comunicaciones, está poblado por redes heterogéneas, donde se convive con redes limitadas en recursos.

La solución en estos casos pasa por escalar las características de seguridad de las redes limitadas NTC a las pasarelas que sirven de interconexión con otras redes en el modelo IoT. Así pues, para ocultar la heterogeneidad de una NTC las pasarelas deben proporcionar las siguientes funcionalidades:

- Adaptación de protocolos entre redes (por definición).
- Creación de túneles entre ellas mismas y otros nodos del dominio NTU (opcional, con

impacto en la evaluación de confianza).

- Gestión de las características de seguridad propias de la red periférica (opcional).
- Descripción de las opciones de seguridad relativas al tráfico originado por un nodo conectado a la pasarela. (Autenticación del nodo de origen, robustez de cifrado, etc.)
- Filtrado del tráfico entrante según políticas de red y de usuario, y las preferencias del nodo de destino (opcional).

Así pues las pasarelas pueden actuar de dos formas: transparente o activa. Una pasarela transparente actúa sólo a nivel de red, transfiriendo y eventualmente encaminando los paquetes entre las redes que emplean diferentes protocolos de nivel de red.

Desde el punto de vista de la seguridad en las redes NTC, se requieren pasarelas activas, que además de las funciones realizadas por las transparentes, sean capaces de actuar en protocolos de nivel superior al de red, añadiendo las funcionalidades extra de seguridad que sean necesarias.

2.1.3.1.5.3. Modelo de privacidad

Al igual que con la seguridad, el modelo de privacidad se describe al margen del modelo de referencia IoT-A en [32]. En resumen, la idea fundamental de este modelo es que las personas y las organizaciones deben estar en control de la información de la que son propietarios, incluyendo aquella que les es inherente por definición. Este control incluye tanto el alcance de la divulgación de la información propia, como el destinatario y el uso que se le pretende dar.

En la Figura 25 se muestra un ejemplo del modelo de privacidad para el caso de un sujeto humano.

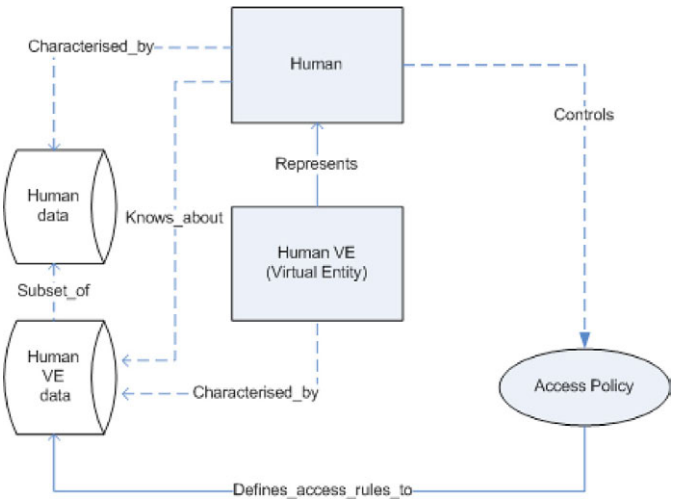

Figura 25. Ejemplo de modelo de privacidad en IoT-A [27]

2.1.3.2. Arquitectura de referencia

La arquitectura de referencia propuesta en el proyecto IoT-A [27] está diseñada como una referencia para la construcción de arquitecturas compatibles con IoT para necesidades específicas. Para su descripción se siguen las definiciones y el catálogo de perspectivas de [34] adaptadas a las necesidades de IoT. Pero sobre todo, y de forma más detallada, se recurre a la descripción de la arquitectura empleando diferentes vistas que permitan comprender mejor su diseño.

Las vistas son representaciones de los aspectos estructurales de la arquitectura de referencia que ilustran como puede ser adoptada para afrontar las preocupaciones de quienes desarrollen arquitecturas basadas en ella para sistemas IoT.

En particular esta arquitectura de referencia se describe en tres vistas: la vista funcional, la vista de información y la vista de despliegue y operaciones. Por relación con este PFC se tratará únicamente la vista funcional.

2.1.3.2.1. Vista funcional

Esta vista emplea componentes funcionales que se agrupan en grupos de funcionalidad en correlación con el modelo funcional ya descrito en el modelo de referencia. El diagrama que describe estos componentes se muestra en la Figura 26, quedando los grupos de aplicación y de dispositivo al margen de la arquitectura de referencia.

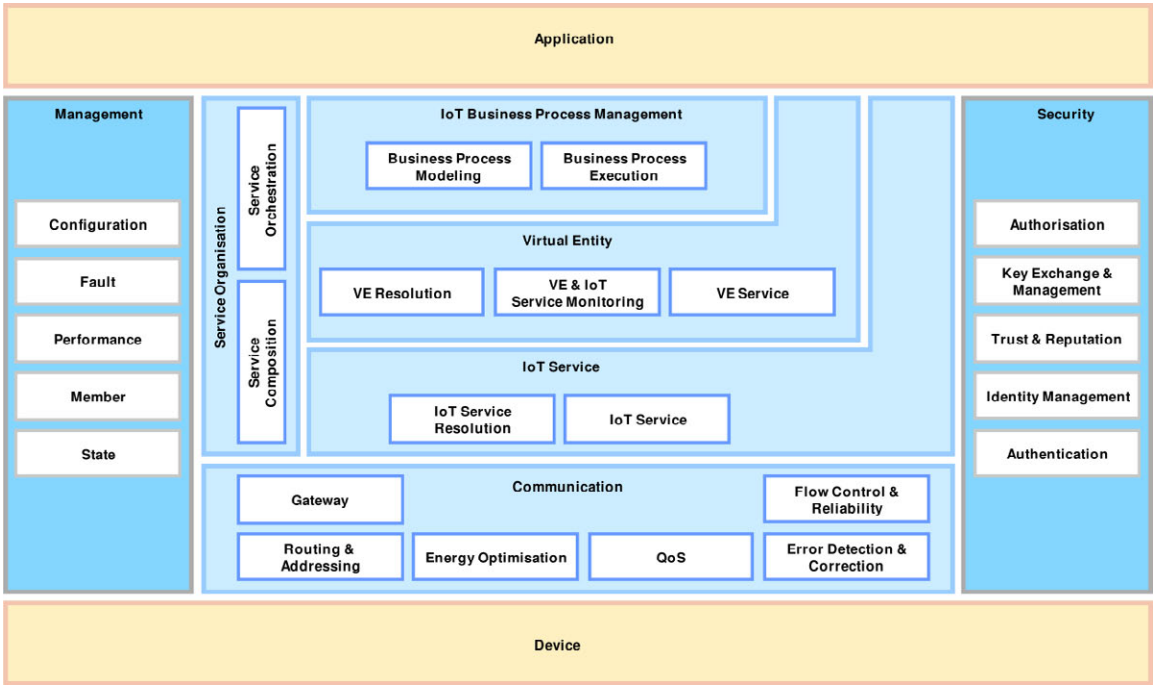

Figura 26. Vista funcional de la arquitectura de referencia IoT-A [27]

2.1.3.2.2.1. Componente de Modelado de Procesos de Negocio

Este componente proporciona el entorno para el modelado de los procesos de negocio IoT que serán serializados y ejecutados en el componente funcional de ejecución de procesos. Sus funciones básicas se describen en la Tabla 4.

Función	Descripción
Modelador de procesos de negocio IoT	Proporciona las herramientas necesarias para el modelado de procesos de negocio empleando una notación normalizada (pendiente de desarrollo por parte de IoT-A).

Tabla 4. Funciones del Componente de Modelado de Procesos de Negocio

2.1.3.2.2.2. Componente de Ejecución de Procesos de Negocio

Ejecuta los procesos de negocio IoT que han sido modelados en el componente de modelado de procesos de negocio. Esta ejecución se lleva a cabo empleando Servicios IoT que son orquestados en la capa de Organización de Servicios. Sus funciones básicas se describen en la Tabla 5.

Función	Descripción
Despliegue de modelos de proceso a los entornos de ejecución	Las actividades de los modelos de proceso IoT se aplican a los entornos de ejecución apropiados, que son los que realizan la ejecución real mediante la localización y la invocación de los adecuados Servicios IoT.
Relación de los requerimientos de aplicación con las capacidades de servicio	Para la ejecución de las aplicaciones, se deben resolver los requerimientos de los Servicios IoT antes de que sean invocados los servicios específicos.
Ejecución de la aplicación	Tras resolver los Servicios IoT, se invocan los servicios correspondientes. La invocación de un servicio es un paso progresivo hacia la ejecución del proceso.

Tabla 5. Funciones del Componente de Ejecución de Procesos de Negocio

2.1.3.2.2.3. Componente de Orquestación de Servicios

Este componente resuelve los Servicios IoT que son los adecuados para cumplir las peticiones de servicio procedentes del componente de Ejecución de Procesos de Negocio, o de otros usuarios IoT-A. Sus funciones básicas se describen en la Tabla 6.

Función	Descripción
Orquestar Servicios IoT	Esta función resuelve los servicios apropiados que son capaces de manejar las peticiones de un usuario IoT. Si es necesario se crearán recursos temporales para almacenar los resultados intermedios que se empleen en la composición del servicio o en el procesado de eventos complejos.

Tabla 6. Funciones del Componente de Orquestación de Servicios

2.1.3.2.2.4. Componente de Composición de Servicios

Este componente resuelve los servicios que están compuestos tanto de Servicios IoT como de otros servicios con la finalidad de crear nuevos servicios con funcionalidades extendidas. Sus funciones básicas se describen en la Tabla 7.

Función	Descripción
Soporte de composiciones Servicios flexibles	Proporciona la resolución dinámica de servicios complejos, compuestos por otros servicios. Estos servicios componibles son elegidos en función de su disponibilidad y de los derechos de acceso del usuario solicitante.
Aumento de la calidad de la información	Esta función puede emplearse para mejorar la calidad de la información mediante la combinación de la información procedente de diferentes fuentes. Por ejemplo el cálculo de un valor medio calculado en los valores facilitados por múltiples fuentes.

Tabla 7. Funciones del Componente de Composición de Servicios

2.1.3.2.2.5. Componente de Resolución de Entidades Virtuales

La resolución de Entidades Virtuales le proporciona al usuario IoT las funcionalidades para obtener las asociaciones entre Entidades Virtuales y Servicios IoT. Quien solicite esta resolución debe poseer al menos las funcionalidades de descubrimiento y búsqueda. Sus funciones básicas se describen en la Tabla 8.

2.1.3.2.2.6. Componente de Entidad Virtual y Monitorización de Servicios IoT

Este componente es el responsable de encontrar automáticamente nuevas asociaciones, que son introducidas en el componente de Resolución de Entidades Virtuales. Las nuevas asociaciones pueden derivarse en función de asociaciones existentes, descripciones de servicio e informaciones acerca de las Entidades Virtuales. Sus funciones básicas se describen en la Tabla 9.

Función	Descripción
Descubrir servicios relacionados con la Entidad Virtual	Descubre, habitualmente de forma dinámica, las asociaciones entre una Entidad Virtual y sus servicios asociados. Para el descubrimiento pueden tenerse en cuenta filtros como la localización, la proximidad, y otra información de contexto. Si no existen asociaciones, pueden crearse.
Suscribir/cancelar la notificación del descubrimiento de asociaciones	Suscribe o cancela la notificación continua al usuario de las asociaciones que se ajusten a las especificaciones suministradas. Cuando un usuario se suscribe recibe un identificador único, que se usa para identificar las notificaciones y para realizar la cancelación de la suscripción.
Búsqueda de servicios relacionados.	Busca por servicios que exponen los recursos asociados a una Entidad Virtual.
Suscribir/cancelar la búsqueda de asociaciones	Suscribe o cancela las notificaciones al usuario basadas en la identidad de la Entidad Virtual y la especificación del servicio. Cuando un usuario se suscribe recibe un identificador único, que se usa para identificar las notificaciones y para realizar la cancelación de la suscripción.
Insertar asociación	Introduce una nueva asociación entre una Entidad Virtual y los Servicios IoT que tiene asociados.
Borrar asociación	Borra una asociación entre una Entidad Virtual y los Servicios IoT que tiene asociados.
Actualizar asociación	Actualiza las asociaciones entre una Entidad Virtual y los Servicios IoT que tiene asociados.

Tabla 8. Funciones del Componente de Resolución de Entidades Virtuales

Función	Descripción
Asociación de afirmación estática	Crea una nueva asociación estática entre Entidades Virtuales y los servicios descritos por la asociación proporcionada.
Asociación de descubrimiento dinámico	Crea una nueva asociación dinámica entre Entidades Virtuales y los servicios descritos por la asociación.
Asociación caducada	Borra una asociación de la Resolución de Entidades Virtuales.
Actualizar Asociación	Actualiza la asociación.

Tabla 9. Funciones del Componente VE y Monitorización de Servicios IoT

2.1.3.2.2.7. Componente de Entidad Virtual de Servicio

Una Entidad de servicio representa el punto de acceso a una entidad particular, ofreciendo los medios para conocer y manipular su estado. Las Entidades de servicio proporcionan accesos a una entidad mediante operaciones que permiten la lectura y la actualización de los valores de los atributos de la entidad. El tipo de acceso a un atributo particular depende de sus características propias. Sus funciones básicas se describen en la Tabla 10.

Función	Descripción
Leer valor de atributo	Devuelve el valor de un atributo de la entidad.
Establecer valor de atributo	Establece el valor de un atributo de la entidad.

Tabla 10. Funciones del Componente de Entidad Virtual de Servicio

2.1.3.2.2.8. Componente de Resolución de Servicios IoT

La Resolución de Servicios IoT proporciona todas las funcionalidades requeridas por el usuario para buscar y ser capaz de contactar con los Servicios IoT. También le da a los Servicios la capacidad de manejar sus descripciones de servicio, para que puedan ser buscadas y descubiertas por el Usuario. Sus funciones básicas se describen en la Tabla 11.

Función	Descripción
Resolver Servicio por ID	Obtiene la dirección de un servicio IoT dada su ID
Suscribir/cancelar la resolución de Servicios por ID	Suscribe o cancela la recepción de notificaciones sobre la resolución de servicios IoT basadas en la ID. Cuando un usuario se suscribe recibe un identificador único, que se usa para identificar las notificaciones y para realizar la cancelación de la suscripción.
Buscar Servicio por ID	Recupera la descripción de un servicio IoT dada su ID.
Suscribir/cancelar la búsqueda de Servicios por ID	Suscribe o cancela las notificaciones continuas acerca de servicios que se justen a la especificación de servicio dada. Cuando un usuario se suscribe recibe un identificador único, que se usa para identificar las notificaciones y para realizar la cancelación de la suscripción.
Actualizar Servicio	Actualiza la entrada de servicio con una nueva descripción.
Insertar Servicio	Añade una nueva entrada de servicio en la base de datos con la descripción facilitada.
Borrar Servicio con ID	Elimina la entrada de un servicio dada su ID.

Tabla 11. Funciones del Componente de Resolución de Servicios IoT

2.1.3.2.2.9. Componente de Servicio IoT

Se trata de un componente software que emplea una interfaz conocida para exponer un Recurso con el objetivo de hacerla accesible a otras partes del sistema IoT, con frecuencia a través de Internet. Los servicios de recurso exponen las funcionalidades de un dispositivo, típicamente accediendo a los recursos que éste tenga alojados. En estas condiciones este tipo de servicios suele hacer referencia a un único recurso por servicio.

También pueden tratarse de aspectos no funcionales, como la fiabilidad, la capacidad de recuperación y el rendimiento.

La arquitectura de referencia no define un conjunto de funciones básicas para este componente.

2.1.3.2.2.10. Componente de Pasarela

El objetivo de este componente es servir de puente entre diferentes redes. Puede hacer frente a diferentes niveles de la red, como se trató anteriormente en el modelo de comunicaciones. Le da al dispositivo que lo implemente la capacidad de actuar como punto de entrada a otra red. Sus funciones principales son las traducciones de protocolo y de direccionamiento necesarias para traspasar las fronteras entre redes. El conjunto de funciones básicas descrito en la referencia se describe en la Tabla 12.

Función	Descripción
Reenviar	Realiza el reenvío de paquetes.
Seguimiento de Conexión / Agregación	Esta función maneja varios paquetes/mensajes a la vez, manteniendo un registro de estado entre recepciones. Los paquetes tienen una relación entre ellos y/o son agregados.
Filtrar	Esta función filtra los paquetes/mensajes analizando sus cabeceras o contenidos.

Tabla 12. Funciones del Componente de Pasarela

2.1.3.2.2.11. Componente de Control de Flujo y Confiabilidad

Este componente hace frente a las necesidades de confiabilidad y control de flujo. Puede desplegarse en el nivel de enlace, en el de transporte, en el de aplicación, o en la propia aplicación. Sus funciones básicas se describen en la Tabla 13.

Función	Descripción
Conectar	Esta función enlaza un destino con una fuente.
Control E/S	Esta función habilita la exposición de opciones del canal o la conexión.
Enviar mensaje	Sean mensajes con o sin confirmación, necesitan ajustarse a un control de flujo.
TX	Esta función envía a ciegas un paquete/mensaje. Se pueden construir otras sobre ella.
RX	Esta función recibe a ciegas el paquete/mensaje. Se pueden construir otras sobre ella.

Tabla 13. Funciones del Componente de Control de Flujo y Confiabilidad

2.1.3.2.2.12. Componente de Encaminamiento y Direccionamiento

Este componente es el que capacita a los nuevos dispositivos la entrada en la red, la obtención de una red y la capacidad de ser alcanzables. De entre las funciones que se le describen, las más importantes son la asignación de direcciones, el mantenimiento de las tablas o políticas de encaminamiento y el reenvío de paquetes de datos. La totalidad de las funciones básicas que se recogen en la arquitectura de referencia para este componente se describen en la Tabla 14.

Función	Descripción
Control de Direcciones	Esta función habilita el descubrimiento, la oferta, la petición y el reconocimiento de direcciones de acuerdo a un método elegido.
Control de Rutas	Esta función habilita la purga, borrado y añadido de rutas.
Suscripción a Información Vecino	Esta función proporciona la suscripción o vigilancia de la información de encaminamiento vecino.
Obtener Grafo	Obtiene el grafo de la ruta.
Control de Información de Encaminamiento	Esta función facilita la Solicitud de Información o el Anuncio de Destino de acuerdo a un método elegido.
Calcular Rango	Calcula el rango de los pares que forman parte de una comunicación.

Tabla 14. Funciones del Componente de Encaminamiento y Direccionamiento

2.1.3.2.2.13. Componente de Optimización Energética

Este componente es el encargado de gestionar el consumo de energía durante el uso de las comunicaciones. Suele implementarse mediante el apagado programado y selectivo del equipo de

transmisión. Habitualmente se implementa en el nivel más bajo de la pila de protocolos, pero puede hacerse en cualquiera de ellos. Las funciones básicas se describen en la Tabla 15.

Función	Descripción
Dormir	Informa al par de comunicación que el nodo se va a poner en modo dormir por un periodo de tiempo. Opcionalmente puede informar de cuándo volverá a estar disponible.
Despertar	Informa al par de comunicación de que el nodo se ha despertado/está disponible. Opcionalmente puede informar de cuando se pondrá a dormir de nuevo.
Control de Radio	Esta función permite encender y apagar la radio.
Comprobar Intervalo	Esta función permite conocer el intervalo dado entre eventos relativos a la comunicación.

Tabla 15. Funciones del Componente de Optimización Energética

2.1.3.2.2.14. Componente de Calidad del Servicio (QoS)

Este componente hace tan sólo referencia a la calidad de servicio aplicada a los servicios de comunicación. En particular se habla de rutas rápidas, latencias, prioridad de paquetes, etc. Los sistemas en tiempo real necesitan interactuar con este componente. Sus funciones básicas se describen en la Tabla 16

Función	Descripción
Asignar Clase de Tráfico	Asigna un canal de comunicación para una clase de tráfico.
Reservar	Reserva un canal de comunicación para una clase de tráfico.

Tabla 16. Funciones del Componente de Calidad de Servicio (QoS)

2.1.3.2.2.15. Componente de Detección y Corrección de Errores

La detección de errores se encuentra presente en diferentes niveles de la pila de protocolos, y puede enfocarse de forma distribuida si se realiza desde los niveles superiores.

La corrección de errores no es parte integral de IoT, pero debe ser considerara importante para mantener la coherencia de las comunicaciones y la topología de la red durante la ocurrencia de problemas transitorios.

Sus funciones básicas se describen en la Tabla 17.

Función	Descripción
Calcular firma	Calcula la firma de una secuencia de acuerdo a un algoritmo. Se suele emplear para firmar un paquete, y según el algoritmo que se emplee podría ser también empleada para verificar la firma.
Verificar firma	Verifica que la firma presente en un paquete corresponde con los datos adjuntos.
Informar Error	Informa de un error.
Sincronizar Tiempo	Proporciona la sincronización entre los relojes internos de los nodos.

Tabla 17. Funciones del Componente de Detección y Corrección de Errores

2.1.3.2.2.16. Componente de Autorización

El componente de autorización es la parte frontal para la gestión de las políticas y la toma de decisiones de control de acceso basadas en las políticas asociadas. Este control de acceso puede ser invocado cada vez que se solicita el acceso a un recurso restringido.

Sus funciones básicas se describen en la Tabla 18.

Función	Descripción
Autorizar	Determina si la acción está autorizada o no en función de la afirmación de identidad, la descripción del servicio y el tipo de acción solicitada.
Gestionar Políticas	Añade, actualiza o borra una política de acceso.

Tabla 18. Funciones del Componente de Autorización

2.1.3.2.2.17. Componente de Autenticación

Este componente está implicado en la autenticación de usuarios y servicios. En él se verifican las credenciales facilitadas por el usuario y, si son válidas, devuelve una afirmación de identidad como resultad, que le será requerida para usar un servicio IoT como cliente. Asimismo cuando se uno un nuevo nodo a la red verifica sus credenciales, y si son correctas establece un contexto seguro entre este nuevo nodo y el resto de entidades en el entorno local. Sus funciones básicas se describen en la Tabla 19.

Función	Descripción
Autenticar	Autentica al usuario en función de las credenciales proporcionadas.
Verificar	Verifica si la afirmación de identidad de un usuario es válida.

Tabla 19. Funciones del Componente de Autenticación

2.1.3.2.2.18. Componente de Gestión de Identidades

Este componente aborda las cuestiones de privacidad de los sujetos de confianza mediante el uso y gestión de seudónimos e información adicional, para que así puedan operar anónimamente. Sus funciones básicas se describen en la Tabla 20.

Función	Descripción
Crear Identidad	Crea una identidad ficticia, así como las credenciales de seguridad relacionadas, para que los usuarios y servicios puedan utilizarla en el proceso de autenticación.

Tabla 20. Funciones del Componente de Gestión de Identidades

2.1.3.2.2.19. Componente de Gestión e Intercambio de Claves

Este componente interviene en el establecimiento de comunicaciones seguras entre dos o más pares IoT-A que no tienen conocimiento previo entre sí.

Sus funciones básicas se describen en la Tabla 21.

Función	Descripción
Distribuir claves de forma segura	Bajo petición, esta función localiza el marco de seguridad común soportado para en el nodo de origen y en el de destino, crea una clave o par de claves en este marco común y las distribuye de forma segura.
Registrar las capacidades de seguridad	Esta función es invocada por los nodos y pasarelas que quieren beneficiarse de la mediación del componente para el establecimiento de conexiones seguras. De esta forma se registran las capacidades que pueden proporcionar las claves necesarias en el marco seguro correcto.

Tabla 21. Funciones del Componente de Gestión e Intercambio de Claves

2.1.3.2.2.20. Componente de Arquitectura de Confianza y Reputación

Este componente recoge información sobre los registros de reputación de los usuarios y calcula los niveles de confianza del servicio. Sus funciones básicas se describen en la Tabla 22.

2.1.3.2.2.21. Componente de Gestión de Fallos

Este componente tiene por objetivo la identificación aislamiento, corrección y registro de los fallos que ocurran en el sistema IoT. Cuando un fallo ocurre en otro componente de la arquitectura, se debe notificar a éste, que será el que realice las acciones que se estimen oportunas para

identificar la naturaleza y gravedad del problema, así como su manejo. Sus funciones básicas se describen en la Tabla 23.

Función	Descripción
Solicitar Información de Reputación	Esta función proporciona el valor de la reputación de una entidad solicitada en el contexto especificado.
Proporcionar Información de Reputación	Esta función es empleada por una entidad remota para proporcionar su información de reputación acerca de otra entidad.

Tabla 22. Funciones del Componente de Arquitectura de Confianza y Reputación

Función	Descripción
Detectar Error	Pregunta a los componentes del sistema por mensajes de error.
Manejar Error	Analiza un fallo y, si es solicitado, inicia la secuencia de acciones para hacerle frente.
Registrar Error	Genera un registro de fallos
Recuperar	Regresa el sistema a su estado anterior.
Manejar Alarma	Reacciona ante una alarma generada por otro componente.
Generar Alarma	Genera una alarma que es propagada a otros componentes.
Estadísticas de Error	Generar estadísticas de fallos usando los datos del registro.

Tabla 23. Funciones del Componente de Gestión de Fallos

2.1.3.2.2.22. Componente de Gestión de Configuración

Este componente inicializa la configuración del sistema, recupera y almacena la configuración de otros componentes funcionales y dispositivos, realiza un seguimiento de los cambios de configuración y planifica las futuras extensiones del sistema. Sus funciones básicas están descritas en la Tabla 24.

Función	Descripción
Inicialización y Cambio	Inicializa o cambia la configuración del sistema.
Autodescubrimiento	Descubre la configuración del sistema.
Respaldo y Recuperación	Realiza o recupera una copia de respaldo de la configuración.
Inventario	Registro de configuraciones incluyendo descripciones asociadas.

Tabla 24. Funciones del Componente de Gestión de Configuración

2.1.3.2.2.23. Componente de Gestión de Rendimiento

Este componente determina la eficiencia del sistema actual mediante la recolección y análisis de datos de rendimiento. Con el establecimiento de marcas o valores es posible predecir situaciones futuras del sistema. Sus funciones básicas se describen en la Tabla 25.

Función	Descripción
Obtener Estado	Descubre el estado del sistema.
Generar Informe	Genera un informe acerca del rendimiento del sistema.
Comportamiento	Aplica un comportamiento particular al sistema. Genera la secuencia de comandos a ser enviados a otros componentes funcionales y ejecuta la lista.
Prueba de Consistencia	Verifica la consistencia de la lista de comandos generada por Comportamiento.

Tabla 25. Funciones del Componente de Gestión de Rendimiento

2.1.3.2.2.24. Componente de Gestión de Miembros

Este componente se encarga de gestionar la afiliación y la información asociada de cualquier entidad relevante a un sistema IoT. Sus funciones básicas se muestran en la Tabla 26.

Función	Descripción
Insertar Miembro	Registra un miembro en la base de datos de afiliaciones. Esta acción activa un intercambio con el grupo funcional de seguridad para verificar si la entidad está autorizada para unirse al sistema.
Borrar Miembro	Elimina un miembro de la base de datos de afiliaciones.
Actualizar Miembro	Actualiza los metadatos de un miembro en la base de datos de afiliaciones.
Seleccionar Miembro	Permite la obtención de un miembro del sistema que cumpla un filtro dado.
Suscribir a Actualizaciones de Afiliación	Permite la suscripción a las actualizaciones de la tabla de afiliaciones que se ajusten a un filtro especificado.
Dar de baja	Darse de baja de las actualizaciones.

Tabla 26. Funciones del Componente de Gestión de Miembros

2.1.3.2.2.25. Componente de Gestión de Estado

El componente de Gestión de Estado se encarga de controlar y predecir el estado de un sistema IoT. Para obtener un diagnóstico rápido del sistema se cuenta con información de su pasado, presente y predicción de futuro. Esta funcionalidad también puede aplicarse para la facturación de servicios. Sus funciones básicas se describen en la Tabla 27.

Función	Descripción
Leer Estado	Obtiene el estado del sistema.
Escribir Estado	Cambia o crea una entrada de estado.
Predecir Estado	Realiza una predicción del estado del sistema para un momento dado.
Suscribir a Actualizaciones de Estado	Suscribe a las actualizaciones de estado de acuerdo al filtro proporcionado.
Dar de baja	Da de baja de las actualizaciones de estado.

Tabla 27. Funciones del Componente de Gestión de Estado

2.1.4. Web de las Cosas (WoT)

La visión de IoT puede generalizarse como la interconexión de una innumerable cantidad de dispositivos empleando Internet para comunicarse e intercambiar información. Parece lógico pensar que paralelamente a esta visión se pueden construir otras empleando tecnologías ya conocidas en el dominio de Internet, como son las tecnologías web.

Web de las Cosas (WoT, por *Web of Things*) recoge precisamente esta nueva visión, en la que los diferentes dispositivos conectados a Internet hacen uso de las normas abiertas que regulan la Web para acometer precisamente la comunicación y el intercambio de información, facilitando la interoperatividad entre ellos y los servicios que provean.

De esta forma las cosas inteligentes (por *smart things*) conectadas a la red pueden actuar como servidores web que ofrecen sus funcionalidades como servicios red directamente accesibles en la web, empleando tecnologías conocidas como SOAP y RESTful, empleando protocolos conocidos como HTTP o CoAP, describiendo sus características con lenguajes de marcado normalizados como DPWS (*Device Profile for Web Services*) y sus servicios con WSDL (*Web Services Description Language*), e intercambiando información empleando XML y JSON para representar los datos.

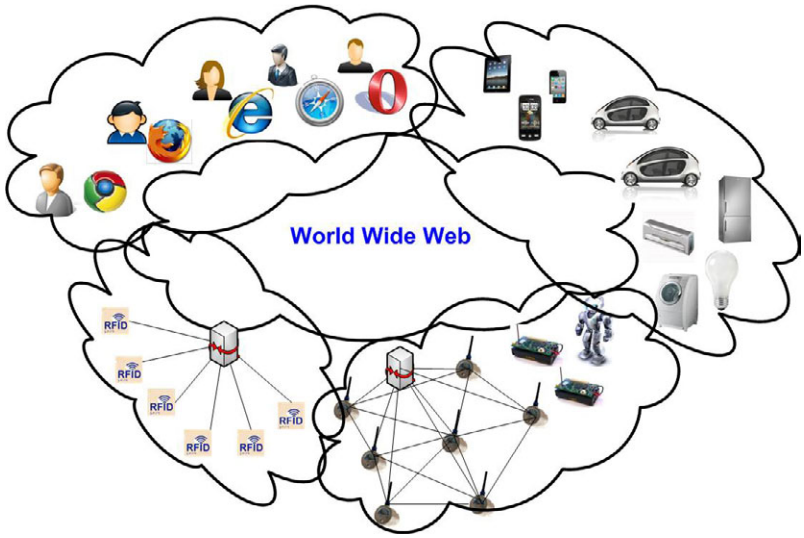

Figura 27. Visión de la Web de las Cosas [35]

Tal y como se muestra en la Figura 27, los dispositivos pueden integrarse en la WoT de dos formas diferentes:

- **Integración directa:** Para integrar las cosas directamente en la Web su primer requisito es que deben ser direccionables en Internet. Es decir, deben tener una dirección IP. Además WoT requiere que exista conectividad e interoperatividad a nivel de aplicación. Esto implica que este tipo de dispositivos debe disponer de al menos un servidor web básico mediante el que pueda ofrecer sus servicios utilizando una interfaz web normalizada. De esta forma se habilita la posibilidad de que las personas puedan acceder a los servicios así expuestos desde cualquier navegador, y a que otras aplicaciones puedan interactuar con ellos mediante protocolos conocidos y normalizados. Existen varias experiencias al respecto, como la realizada por Akribopoulos y su grupo de la Universidad de Patras [36], empleando dispositivos SunSPOT para exponer sus recursos directamente en la Web como servicios web.
- **Integración indirecta:** Existen dispositivos que por sus propias características es posible que no puedan ser direccionables en Internet de forma directa, ni mucho menos ofrecer un servidor web capaz de proporcionar la conectividad necesaria a nivel de aplicación, como por ejemplo las etiquetas RFID. En otros casos es posible que no exista la necesidad de realizar una integración directa en beneficio del coste, el ahorro energético y la seguridad, como por ejemplo en los nodos de una red de sensores y actuadores. En estos casos se recurre a un elemento intermedio entre las cosas y la Web, al que suele referirse como *Gateway*, puerta de enlace o pasarela.

2.2. Comunicaciones de nivel físico a nivel de red

El diseño y construcción de un sistema IoT acorde a la arquitectura de referencia que se describe en la sección anterior permite el uso de una amplia diversidad de tecnologías de red. Desde el nivel físico hasta el nivel de información existen varias opciones tecnológicas que deben tenerse en consideración según las redes y entidades participantes que vayan a utilizarse.

Ya se describió en la Tabla 3 la clasificación de los tipos de red que se emplean en un sistema IoT. De igual forma se describieron los tipos de terminales o dispositivos en la relación entre el modelo de comunicaciones y el de información que se resume en la Figura 23. Relación entre los modelos de comunicación y de información en IoT.

Adicionalmente en la propuesta de protocolo para IoT en IoT-A [37] se describen las funcionalidades completas que se pueden dar en una pasarela IoT, añadiendo a las tratadas con anterioridad la de Registro de Dispositivos y Servicios. Se trata de un registro central que mantiene una lista de los servicios y recursos disponibles en el dominio, y que puede ubicarse en la propia pasarela, en un dispositivo dedicado, o en un servicio que resida en una red sin limitaciones NTU.

Esta propuesta aborda diferentes escenarios posibles de comunicación, analizando las opciones para cada nivel de la pila de protocolos propuesta. En particular se establece una relación con las comunicaciones M2M, al punto de considerar que las comunicaciones en el nivel de información deben ser compatibles con éste.

En las siguientes subsecciones se van a describir los tipos de comunicación que se consideran dentro de un sistema IoT, las topologías de red que pueden encontrarse en este tipo de sistemas, y los protocolos y tecnologías de red en uso en los niveles inferiores al correspondiente a la comunicación extremo a extremo. Todo ello centrado en las NTC, y especialmente en las WSAN, dado que son este tipo de redes las que son objeto de este PFC.

2.2.1. Tipos de comunicaciones

En un sistema IoT las comunicaciones deben ser de alguno de los tipos que se definen a continuación:

- **Unicast**: Es la comunicación directa entre dos terminales.
- **Multicast**: Es cuando un terminal envía información a un grupo de terminales que

pueden encontrarse en el mismo dominio de red o en otro diferente.

- **Multicast Inverso**: Un grupo de terminales envía información, posiblemente agregada en un punto intermedio, a un único terminal ubicado en el mismo dominio de red o en otro diferente.
- **Anycast**: Un único terminal de origen envía información a un terminal de destino que es seleccionado en función de un conjunto de atributos especificados por el remitente en lugar de por la identidad o la dirección de red del destinatario.
- **Broadcast**: Un único terminal envía la información a todos los terminales de una red. Este tipo de comunicación se puede realizar en los niveles de enlace y de red. En algunos casos puede especificarse la distancia a la que puede ser enviada la información.

2.2.2. Topologías de red

Cuando se habla de topologías de red conviene tener presente que existen dos categorías básicas: topologías físicas y topologías lógicas.

Las topologías físicas son las que hacen referencia a la ubicación e interconexión de los nodos que forman parte de una red. Por su parte las topologías lógicas hacen referencia a la forma en la que la información fluye por la red con independencia de su estructura física.

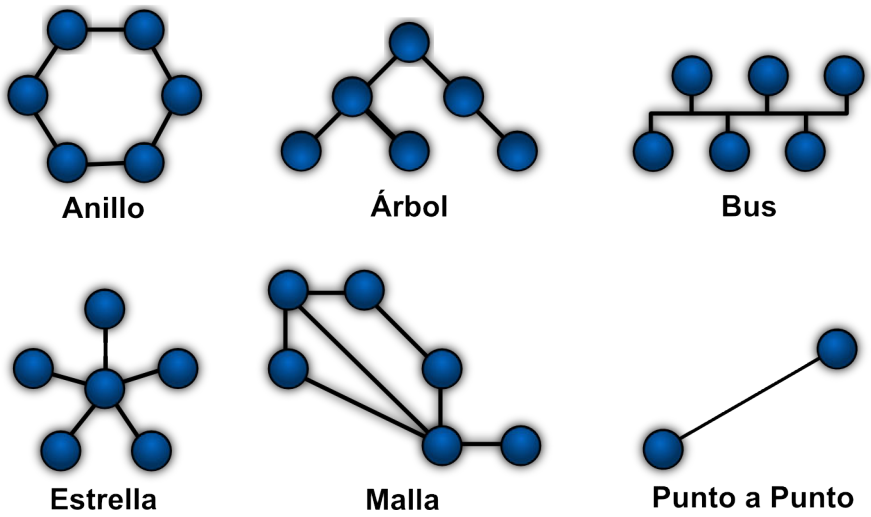

Figura 28. Topologías de red

De las topologías de red que se ilustran en la Figura 28, las WSAN se organizan por naturaleza como una red de malla a nivel físico, en la que todos los nodos que forman parte de la red pueden

físicamente conectarse al resto.

Es a nivel lógico donde una WSAN utiliza diferentes topologías, siendo lo común una hibridación de las ilustradas, de entre las que destacan las topologías *ad hoc* o definidas a propósito para una solución concreta, la superposición de topologías y las organizaciones tipo clúster o por grupos.

2.2.3. Tecnologías

2.2.3.1. 802.15.4

En la actualidad la tecnología de preferencia para los niveles inferiores de la comunicación en una Red Inalámbrica de Área Personal (WPAN) con capacidades limitadas (LoWPAN por sus siglas en inglés) es la propuesta por la norma IEEE 802.15.4. En concreto se define la arquitectura de protocolo para el nivel físico y de enlace (definido como de control de acceso al medio).

A nivel físico 802.15.4 gestiona el transmisor de radiofrecuencia y realiza la selección de canal y las funciones de gestión de energía con el objetivo de optimizarla.

Originalmente se definieron tres bandas de operación en frecuencias que no requieren licencia para trabajar sobre ellas. Para Europa se asignó la banda de 868 MHz con 3 canales, siendo la de 915 MHz con 30 canales asignada a Norte América, y la de 2.4 GHz con 16 canales para su uso a nivel mundial.

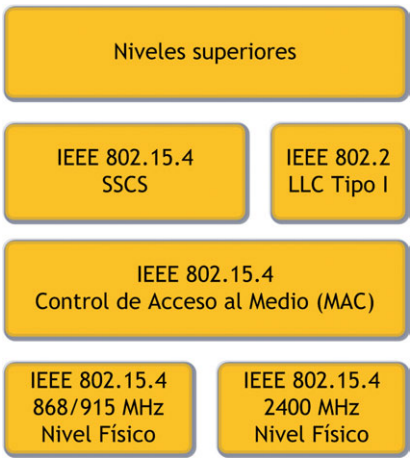

Figura 29. Arquitectura IEEE 802.15.4

Posteriores revisiones de la norma han ido ampliando el número de bandas añadiendo en 2009 las bandas de 780 MHz y 950 MHz.

Las tasas de transferencia varían entre los 20 Kbit/s para la banda de los 686 MHz y los 250 Kbit/s para la banda de 2.4 GHz, y el alcance de la transmisión oscila entre los 10 y los 30 metros.

A nivel físico la comunicación se realiza mediante el envío de paquetes que siguen la estructura ilustrada en la Figura 30.

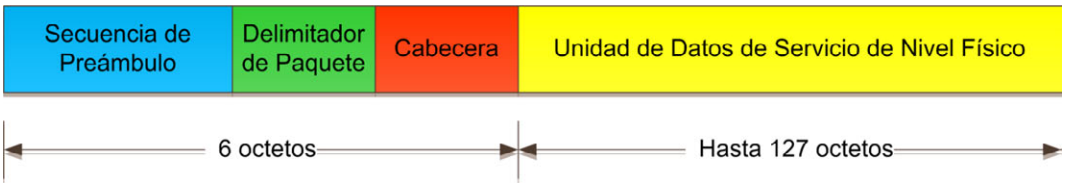

Figura 30. Estructura del paquete de nivel físico en IEEE 802.15.4

El paquete comienza con un campo de preámbulo de 32 bits empleado para la sincronización de símbolos. Le sigue el delimitador del inicio del paquete con 8 bits y la cabecera propiamente dicha con otros 8 bits que contienen información acerca del tamaño de la unidad de datos que va a continuación.

A nivel de enlace se utilizan tramas de tamaño variable que pueden componerse en supertramas de hasta 16 tramas. La estructura de una trama general se ilustra en la Figura 31, siendo los números que aparecen en la parte superior el tamaño de cada campo en octetos.

Figura 31. Estructura de una trama MAC en IEEE 802.15.4

2.2.3.2. Bluetooth

Bluetooth es una especificación industrial para WPAN que facilita la transmisión de voz y datos entre diferentes dispositivos empleando un enlace por radiofrecuencia en la banda de 2.4 GHz. Gestionado por el Grupo de Especial Interés (SIG) formado en 1998 por Ericsson, Nokia, IBM, Toshiba e Intel, se han ido publicando diferentes revisiones y actualizaciones de la especificación, de las cuales tan sólo la primera fue reconocida como norma IEEE bajo la designación IEEE 802.15.1.

Para mejorar las tasas de transferencia de datos Bluetooth emplea en su última versión un enlace sobre 802.11 (Wi-Fi), mientras que el enlace Bluetooth propiamente dicho se emplea para la negociación y el establecimiento de la conexión. De esta forma se pueden alcanzar hasta 24 Mbit/s

en la transferencia de datos, manteniendo el resto de comunicaciones en canales de inferior tasa de transferencia.

Para una mayor información sobre Bluetooth se recomienda consultar [31].

2.2.3.3. ZigBee

ZigBee es una especificación para un conjunto de protocolos de alto nivel usando dispositivos de radio digitales pequeños y de baja potencia que funcionen sobre IEEE 802.15.4 para WPAN. Como tal opera en la misma banda y se sitúa en los niveles superiores de la pila de protocolos ofreciendo directamente un nivel de red y sobre él un nivel de aplicación.

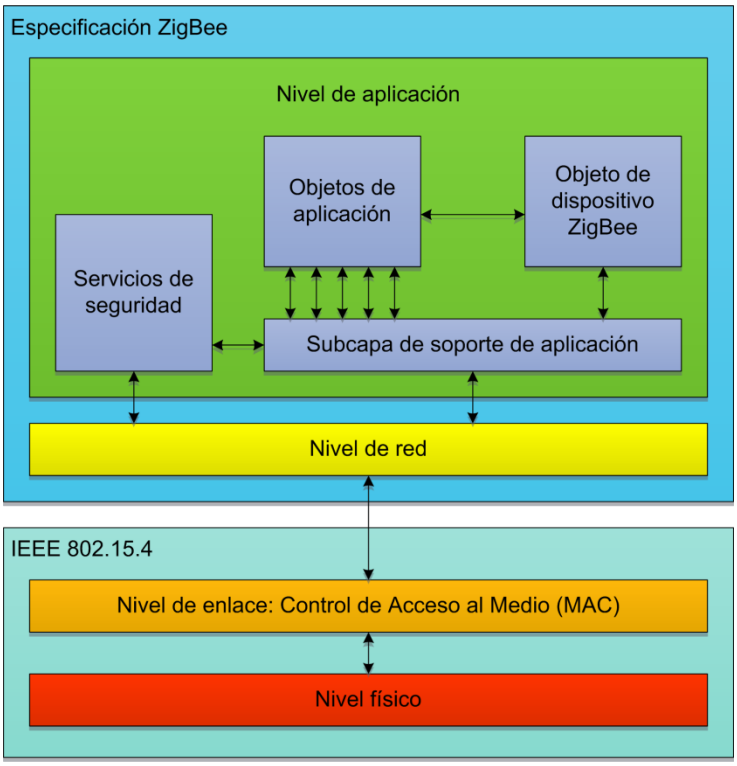

Figura 32. Estructura de la especificación ZigBee

ZigBee está desarrollado por la Alianza ZigBee [38], que es una agrupación de compañías formada en 2002 para mantener y desarrollar la norma. El término ZigBee es una marca registrada además de una norma técnica.

La especificación está disponible gratuitamente para particulares bajo solicitud siempre y cuando no se haga un uso comercial de la misma. La solicitud implica la inscripción como miembro de la alianza en calidad de adoptante de la tecnología, suponiendo graves conflictos a la hora de emplearla en soluciones basadas en licencias libres del tipo de la Licencia Pública General de GNU.

Esta situación ha llevado a los desarrolladores de sistemas libres a considerar el abandono de ZigBee a favor de otras tecnologías normalizadas. En particular, existe un esfuerzo para llevar las tecnologías TCP/IP a las redes LoWPAN, por lo que el salto hacia su uso parece el siguiente paso lógico.

2.2.3.4. 6lowPAN

La especificación base para 6LoWPAN ha sido desarrollada por el grupo de trabajo del mismo nombre del IETF en la forma de la recomendación RFC 4944 [39] atendiendo a los problemas que se describen en la RFC 4919 [40], y siendo actualizada con una mejora en el mecanismo de compresión de cabeceras en la RFC 6282 [41].

Las características que definen una red LoWPAN, y que son tenidas en cuenta a la hora de desarrollar 6LoWPAN son:

- Paquetes de pequeño tamaño. Como se ha visto en la descripción de IEEE 802.15.4, el tamaño máximo de la información que puede contener un paquete es de 127 octetos, que en realidad se convierten en un máximo de 102 al introducir las tramas del nivel de enlace. Y si además se emplean mecanismos de seguridad, esta cifra se reduce hasta los 81 octetos.
- Soporte para direcciones cortas de 16 bits o extendidas de 64 en el nivel de enlace.
- Reducido ancho de banda con tasas de transferencia máximas de 250 kbit/s y mínimas de 20 Kbit/s.
- Topologías en estrella y malla.
- Poca potencia. Lo habitual es que se traten de dispositivos que funcionan a base de baterías.
- Bajo coste, lo que influye en otras características como baja capacidad de procesamiento y de almacenamiento.
- Esperanza de uso de un gran número de dispositivos, mencionándose cifras del orden de los millardos y superiores.
- La ubicación de los dispositivos no sueles ser conocida por anticipado.
- Los dispositivos de una LoWPAN tienen a no ser confiables, en el sentido de que pueden no estar disponibles por problemas de conectividad, batería, bloqueos, etc.
- En algunos entornos los dispositivos de una LoWPAN se inactivan durante largos periodos de tiempo para ahorrar energía.

El problema más relevante recae sobre el tamaño del paquete de datos. En IPv6 el tamaño máximo de transmisión (MTU) debe ser al menos de 1280 octetos [42], pero como hemos visto, el tamaño máximo disponible en una red LoWPAN basada en IEEE 802.15.4 puede verse limitada a 81 octetos. Si se empleasen las cabeceras normalizadas de IPv6 esta cifra se vería reducida hasta los 33 octetos, lo que resultaría poco práctico.

6LoWPAN resuelve parcialmente este problema empleando técnicas de compresión de las cabeceras que en el mejor de los casos pueden albergar hasta 116 octetos destinados a datos, y al menos 72 en el peor posible.

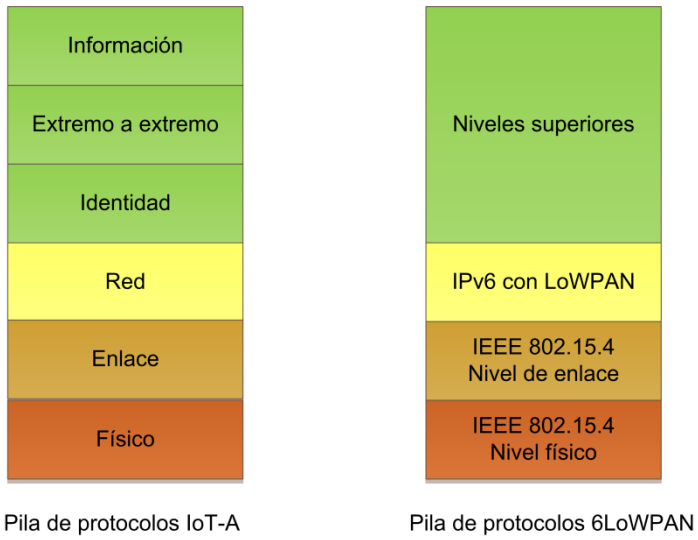

Figura 33. Comparativa de las pilas de protocolos IoT-A y 6LoWPAN

2.3. Tecnologías de comunicación web

2.3.1. Protocolo de transferencia de hipertexto (HTTP)

El protocolo de transferencia de hipertexto HTTP es la base de funcionamiento de la web, y es fruto del esfuerzo de colaboración entre la IETF y el Consorcio para la World Wide Web (W3C), que dio lugar en 1999 a la versión 1.1 del protocolo [43], y que sigue en uso en la actualidad.

HTTP es un protocolo que funciona siguiendo el modelo de peticiones y respuestas dentro de un modelo distribuido basado en clientes y servidores. Los clientes realizan las peticiones empleando unos mensajes normalizados, conocidos como métodos, y que son dirigidos a un recurso alojado en

un servidor e referenciado por un identificador único (URI). El servidor que atiende las peticiones ejecuta las acciones que impliquen el tipo de mensaje empleado, y envía la respuesta correspondiente incluyendo un valor de estado que informa del resultado de la operación sobre el recurso.

En la Tabla 28 se recogen los métodos HTTP descritos en la versión 1.1 del protocolo.

Método	Descripción
OPTIONS	Representa una petición de información acerca de las opciones de comunicación disponibles en la cadena de peticiones y respuestas asociadas por la URI empleada en la petición.
GET	Recupera cualquier información identificada por la URI de la petición. Pueden emplearse condicionales para modificar el tipo de respuesta por ejemplo si la información ha sido modificada desde una fecha indicada.
HEAD	Este método se gestiona de igual forma que GET, con la diferencia de que el servidor no devuelve una respuesta con la información asociada a la URI de la petición, sino que en su lugar envía metainformación en las cabeceras de la respuesta.
POST	Se utiliza para solicitar al servidor que acepte el contenido que se envía junto a la petición como subordinado del recurso indicado por a URI. Se emplea para realizar anotaciones sobre los recursos existentes, publicar mensajes, proporcionar bloques de datos y ampliar el contenido de una base de datos.
PUT	De forma similar a POST, adjunta información con la petición que en el caso de que el recurso exista se emplea para actualizarlo, y en caso contrario el servidor puede crear un nuevo recurso como si hubiera recibido el correspondiente POST.
DELETE	Borra el recurso especificado por la URI de la petición.
TRACE	Obtiene una réplica de la petición para poder ser analizada y contrastada con la original en busca de las modificaciones que hayan podido introducir servidores intermedios.
CONNECT	Convierte la solicitud en un túnel a través de un proxy.
PATCH [44]	Realiza un conjunto de modificaciones descritas en la petición sobre el recurso identificado por la URI.

Tabla 28. Métodos definidos en el protocolo HTTP/1.1

2.3.2. Constrained Application Protocol (CoAP)

El protocolo de aplicación limitado o CoAP es un borrador activo en desarrollo en el IETF para proporcionar un protocolo de nivel de aplicación equivalente a HTTP para redes y dispositivos con recursos limitados, como los que se pueden encontrar en WSAN y otras redes M2M [45] [46].

El grupo de trabajo del IETF ha definido las siguientes características para CoAP:

- Diseño de un protocolo REST con menor complejidad para las asociaciones con HTTP.
- Baja sobrecarga de las cabeceras y de la complejidad de interpretación.
- Admisión de URI e información de tipo de contenido.
- Capacidad para el descubrimiento de recursos proporcionados por servicios CoAP conocidos.
- Suscripciones sencillas a recursos, y a las notificaciones resultantes.
- Almacenamiento temporal de datos simple basado en la edad del recurso.

Adicionalmente a las características descritas, CoAP considera la posibilidad de que los mensajes sean enviados de forma multicast, no soportada inicialmente por el protocolo HTTP.

Como en HTTP, CoAP soporta dos tipos de mensaje: peticiones y respuestas. Ambos usan cabeceras codificadas en binario para reducir su tamaño, puesto que su uso normal será sobre redes IEEE 802.15.4, y los mensajes deberían ajustarse al tamaño de datos de una trama para evitar la fragmentación.

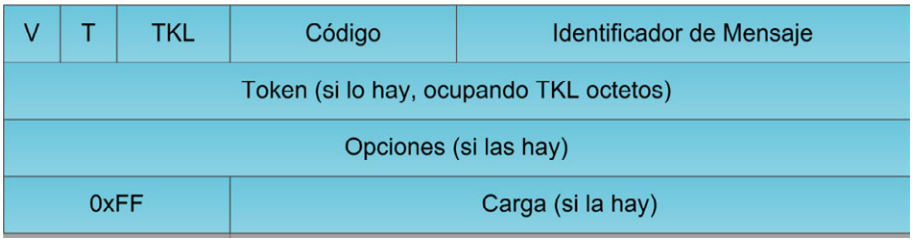

Figura 34. Formato del mensaje CoAP

En la Figura 34 se representa el formato de un mensaje CoAP. El mensaje se inicia con 2 bit sin signo para indicar la versión del protocolo, actualmente sólo contemplada la versión 1. Le siguen otros 2 bit sin signo para el tipo de mensaje, que puede tomar los valores 0 a 3, indicando respectivamente que se trata de un mensaje comprobable, no comprobable, de reconocimiento o de reinicio.

A continuación va el tamaño del token, en 4 bits sin signo, y que puede ser de entre 0 y 8 octetos. Después va un código de mensaje en 8 bits sin signo, cuyos valores pueden ser 1-31 para las peticiones, 64-191 para las respuestas o 0 si va vacío.

El identificador del mensaje ocupa 16 bits sin signo, y se emplea para detectar mensajes duplicados y para emparejar los mensajes de reconocimiento y de reinicio a los mensajes de tipo comprobable y no comprobable.

A la cabecera le sigue un valor de token cuyo tamaño se ha especificado en el correspondiente TKL. El token se emplea para correlacionar peticiones y respuestas.

Tras el token, si las hay, van las opciones. Y finalmente, si hay mensaje de carga, este se incluye en el último lugar precedido de un octeto con todos sus bits puestos a 1.

2.4. Representación de datos y servicios

2.4.1. Extensible Markup Language (XML)

El lenguaje de marcas extensible XML es un lenguaje desarrollado por el W3C con la finalidad de ser empleado en la descripción de datos en formato legible. Deriva del lenguaje de marcas general SGML, permitiendo la definición de una gramática propia para un lenguaje específico.

XML describe una clase de objetos llamados documentos XML y parcialmente describe el comportamiento de los programas de ordenador que pueden procesarlos [47]. Los documentos XML se componen de secuencias de caracteres que pueden pertenecer a dos tipos: marcas o contenidos.

Las marcas son símbolos especiales que indican que el texto que delimitan debe ser procesado de alguna forma. Con ellas se le da una estructura lógica a los documentos XML. Las marcas pueden ser de apertura, de cierre, y autoconclusivas y pueden ser cualquier cadena de texto delimitada por los caracteres ‘<’ y ‘>’. En el caso de las marcas de cierre, el texto que define la marca debe precederse con el carácter ‘/’. El mismo carácter se emplea en las marcas autoconclusivas, pero al final.

Empleando las marcas para definir la estructura del documento XML, éste adopta una forma árbol donde sus elementos se distribuyen jerárquicamente. La raíz de este árbol, y por tanto del documento, es el propio documento en sí mismo.

Existen unas reglas fundamentales a la hora de crear un documento XML, dando lugar a lo que se conoce como documento «bien formado»:

- Un documento XML sólo puede tener un elemento raíz.
- Todos los elementos con contenido deben tener marcas de apertura y de cierre.
- Las marcas de elementos diferentes no pueden solaparse.
- Las marcas anidadas se deben cerrar en el orden inverso al de apertura.
- Los atributos que se incluyan en las marcas deben ir siempre entre comillas simples o dobles.
- Los caracteres '<', '>', '&' y '/' se encuentran reservados.

Todo documento XML consta de dos partes. En la primera parte, o prólogo, se realiza la declaración XML, y se indican las instrucciones de procesamiento junto a la declaración del tipo de documento. La segunda parte es el contenido del documento, compuesto por la raíz y el resto de elementos de la estructura.

Además del concepto de «bien formado«, XML utiliza el de validez de un documento acorde a un esquema dado. Como herencia de SGML el esquema puede ser definido empleando un lenguaje conocido como Definición de Tipo de Documento (DTD), si bien tiene limitaciones que han sido superadas por la recomendación XML Schema del W3C [48].

XML Schema es un tipo de documento XML que puede ser utilizado para expresar un conjunto de reglas a las que debe atenerse un documento XML para ser considerado válido según el citado esquema.

2.4.1.1. Efficient XML Interchange (EXI)

Uno de los principales problemas de XML es que se trata de un formato recargado para la descripción de datos, lo que le hace impracticable para redes y dispositivos con recursos limitados. Conscientes de esta situación, el W3C ha adoptado como recomendación un formato binario para XML destinado al intercambio de información llamado Intercambio Eficiente de XML o EXI por sus siglas en inglés [49].

El formato EXI ha sido diseñado para alcanzar las siguientes características:

- Debe ser de uso general. Uno de los principales objetivos de EXI es maximizar el número de sistemas, dispositivos y aplicaciones que puedan comunicarse empleando datos XML.

- Debe tender a un conjunto mínimo. Para alcanzar el más amplio conjunto de aplicaciones pequeñas, móviles y empotradas es preferible contar con aproximaciones simples y elegantes.
- Debe ser eficiente. EXI tiene que ser competitivo con otros formatos binarios diseñados a mano para que pueda ser empleado en aplicaciones que requieran este nivel de eficiencia.
- Debe ser flexible. Hay que considerar la flexibilidad y la eficiencia con documentos que contengan extensiones arbitrarias del esquema.
- Debe ser interoperable. Debe poderse integrar con facilidad con otras tecnologías XML existentes, minimizando los cambios necesarios en ellas.

Los mecanismos que sigue EXI para reducir el flujo de datos son:

- Codificación optimizada de la estructura XML. La información estructural de un documento XML suele ser muy redundante. Los elementos y los atributos se repiten con frecuencia, y por supuesto, toda marca de cierre es precedida por una marca de apertura equivalente. Este tipo de redundancia se puede comprimir empleando gramáticas que representan las posibles transiciones de estado en cualquier parte del documento, por lo que pueden emplearse sólo unos pocos bits para representar esta información. EXI puede operar en estos casos tanto contando con un esquema que describa la estructura del documento XML, como con documentos XML que carezcan de él.
- Representación eficiente de los datos de contenido. En el caso de contar con un esquema para describir la estructura del documento XML, los datos de contenido están asociados a tipos de datos previamente conocidos. EXI emplea esta información, junto a un conjunto de datos propios equivalentes a los empleados en el esquema, para mejorar la compresión.
- Compresión genérica. Adicionalmente EXI realiza un paso final de compresión genérica.

2.4.2. JavaScript Object Notation (JSON)

JavaScript Object Notation (JSON) es un formato de intercambio de datos ligero basado en texto e independiente del lenguaje [50]. Es un lenguaje fácil de leer y escribir por humanos, y de procesar y generar por máquinas.

Los tipos básicos en JSON son:

- Números, en formato decimal.
- Cadenas de texto, formadas por una secuencia de cero o más caracteres Unicode delimitados por el uso de comillas dobles, y con la posibilidad de utilizar secuencias de escape precediéndolas del carácter '/'.
- Tipos *boolean* o lógicos que pueden tomar los valores *true* o *false*.
- *Arrays*, que son una secuencia ordenada de valores delimitada por el uso de corchetes.
- Objetos, que son colecciones no ordenadas de pares nombre/valor, empleando ':' para separar nombres de valores, separando los pares por comas, y estando delimitados por llaves '{' y '}'.

```
{ "value": 31.5 }
```

Cuadro 1. Ejemplo básico de JSON

2.4.2.1. JSON Schema

Una de las limitaciones más importantes a la hora de emplear JSON para la representación de datos en sistemas de comunicación es la ausencia de un formato normalizado de esquema para definir una estructura que deba ser cumplida en orden de verificar la validez de un documento JSON.

IETF está trabajando en el borrador de una propuesta de esquema conocida como JSON Schema [51]. En ella se proporciona la forma en la que definir un contrato para que los datos JSON requeridos por una aplicación cumplan un patrón predefinido en forma similar a los requerimientos que XML Schema determina sobre los documentos XML para ser válidos de acuerdo al esquema.

```
{
	"name": "Temperature",
	"properties": {
		"value": {
			"type": "number",
			"description": "Value of Temperature",
			"required": true
		}
	}
}
```

Cuadro 2. Ejemplo básico de esquema JSON

2.4.3. Web Service Description Language (WSDL)

El lenguaje de descripción de servicios web WSDL es un lenguaje normalizado por el W3C [52] y definido en XML para, como su propio nombre indica, describir servicios web. Con él se describen la interfaz pública del servicio, las operaciones disponibles y el formato de los mensajes de forma abstracta.

En la Tabla 29 se recogen las marcas XML empleadas para definir un servicio web en WSDL.

WSDL 1.1	WSDL 2.0	Descripción
types	types	Describe los datos empleando el lenguaje XML Schema.
message	-	Define los parámetros de las operaciones del servicio web. Esta marca se ha eliminado en la versión 2.0 de WSDL pasando a emplearse los tipos definidos en XML Schema para definir las entradas, salidas y fallos de las operaciones.
portType	interface	Define el servicio web, las operaciones que pueden ser realizadas y en el caso de WSDL 1.1 los mensajes que están involucrados.
binding	binding	Especifica los protocolos empleados en la invocación al servicio.
operation	operation	Define las acciones y las formas en las que se codifica el mensaje. Es el equivalente a los métodos o las llamadas a función de los lenguajes de programación tradicionales.
service	service	Especifica el conjunto de direcciones de red que dan acceso al servicio.
port	endpoint	Define la dirección o punto de acceso al servicio web.

Tabla 29. Marcas para la definición de servicios en WSDL

2.4.4. Web Application Description Language (WADL)

El lenguaje de descripción de aplicaciones web (WADL) es una propuesta realizada al W3C en 2009 por la antigua compañía de software Sun Microsystems, Inc. con la finalidad de describir aplicaciones web en XML en forma similar a WSDL, pero centrándose en servicios tipo REST [53].

En lugar de mediante operaciones e interfaces como se hace en WSDL, una aplicación web es descrita en WADL mediante el uso de un conjunto de recursos. Cada recurso contiene una serie de elementos de tipo *param*, que son los que describen las entradas, y otros elementos de tipo *method* que hacen referencia al método HTTP empleado para acceder al recurso. Éste elemento a su vez está

compuesto por elementos de tipo *request* para especificar las peticiones, y *response*, para las respuestas recibidas.

El W3C no tiene planes para acoger la propuesta WADL, especialmente tras haber mejorado el soporte para los servicios web basados en REST en la versión 2.0 de WSDL. Sin embargo existen productos y aplicaciones que emplean WADL y WSDL para diferenciar entre tipos de servicio web.

En el Cuadro 3 se muestra como ejemplo la descripción de un servicio REST empleando WADL que es generada automáticamente por una de las plataformas empleadas en este PFC.

```
<application xmlns="http://wadl.dev.java.net/2009/02”
             xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <grammars></grammars>
  <resources base="http://localhost:8181/cxf/7F00.0101.0000.1001">
    <resource path="/">
      <resource path="temperature">
        <method name="GET">
          <request></request>
          <response>
            <representation mediaType="application/xml"/>
          </response>
        </method>
      </resource>
    </resource>
  </resources>
</application>
```

Cuadro 3. Ejemplo de definición de servicios en WADL

2.4.5. Service Mapping Description (SMD)

La descripción de asociaciones de servicio SMD es una propuesta de recomendación que describe en formato JSON los servicios web disponibles en un particular punto de acceso siguiendo un objetivo similar a WSDL [54].

Una SMD se describe en un objeto JSON con propiedades que son empleadas para definir la interacción de los servicios web disponibles. En la raíz la SMD puede contener un servicio o una lista de ellos. Cada uno de los servicios puede tener sus propios atributos, mientras que los que son definidos en la raíz son heredados por todos los servicios de la lista.

2.5. Arquitecturas de desarrollo de aplicaciones web

2.5.1. Simple Object Access Protocol (SOAP)

El protocolo de acceso simple a objetos SOAP proporciona una forma de comunicarse entre aplicaciones que se ejecutan en diferentes sistemas operativos, empleando diferentes tecnologías y lenguajes de programación. Se trata de una recomendación del W3C [55] que sirve para definir el intercambio estructurado de información en la implementación de servicios web empleando XML.

Un mensaje SOAP se compone los cuatro elementos que se describen en la Tabla 30. En la raíz se ubica el sobre o *envelope*, que contiene al resto de elementos posibles.

Elementos	Descripción	Tipo
Envelope	Identifica el documento XML como un mensaje SOAP	Obligatorio
Header	Contiene información	Opcional
Body	Contiene la información de las llamadas y la respuestas	Obligatorio
Fault	Proporciona información de los errores en el procesado del mensaje.	Opcional

Tabla 30. Elementos de un mensaje SOAP

Las principales características de SOAP son:

- **Extensibilidad**, mediante la que se permiten crear nuevas extensiones al protocolo.
- **Neutralidad**, pues si bien se utiliza habitualmente para la distribución de servicios web, puede ser empleado sobre otros protocolos además de HTTP.
- **Independencia,** puesto que se puede emplear con cualquier modelo de programación.

SOAP es por definición un protocolo sin estado, que sigue el paradigma del intercambio de mensajes en una sola dirección, pero en el que las aplicaciones pueden crear patrones de interacción más complejos combinando tales tipos de intercambio.

Uno de sus mayores inconvenientes es que al emplear XML para la construcción de los mensajes intercambiados, dada la naturaleza recargada de este lenguaje, SOAP se puede considerar lento comparado con otras opciones. Sin embargo este problema se puede solucionar empleando mensajes cortos y/o empleando compresión en los extremos.

2.5.2. Representational State Transfer (REST)

La transferencia de estado representacional REST es una arquitectura de software para el desarrollo de aplicaciones web basada en el aprovechamiento de los métodos definidos en el protocolo HTTP, y que fue propuesta originalmente por Roy Fielding en el año 2000 como parte de su tesis doctoral [56].

REST sigue un modelo tradicional cliente/servidor. Pero a diferencia de exportarse servicios mediante la publicación de una interfaz donde se den a conocer los métodos expuestos y la forma de acceder a ellos (como con SOAP y WSDL), las peticiones y respuestas se realizan sobre recursos alojados en el servidor. Además las peticiones emplean métodos de conocida eficacia tomados del protocolo HTTP.

En el contexto de REST un recurso es cualquier cosa que pueda ser direccionable, y las operaciones que el cliente realiza sobre él obtienen como resultado una representación del mismo. De esta forma se evoca la imagen de una máquina de estados, representada por el servidor. Las peticiones del cliente provocan la transición a un nuevo estado, y la respuesta generada por el servidor corresponde a la transferencia del nuevo estado al cliente.

De los métodos descritos en el protocolo HTTP REST emplea fundamentalmente cuatro: GET, POST, PUT y DELETE. Opcionalmente se pueden emplear otros métodos, como PATCH, HEAD y OPTIONS, aunque para ser considerada una implementación REST completa basta con los cuatro principales.

De esta forma con REST se sigue un principio de operaciones CRUD (Create, Read, Update and Delete, o Crear, Obtener, Actualizar y Borrar) básico para el almacenamiento persistente, y que resulta de utilidad en la transferencia de estados. La equivalencia entre los métodos REST y las operaciones CRUD se muestran en la Tabla 31

CRUD	HTTP
Crear	POST
Obtener	GET
Actualizar	PUT / PATCH
Borrar	DELETE

Tabla 31. Equivalencias de operaciones CRUD a métodos HTTP

Una arquitectura REST es lo suficientemente abierta como para que sus componentes se puedan diseñar con total libertad. Aun así hay una serie de restricciones que deben cumplirse para que sea considerada REST completa o RESTful [56]:

- **Cliente – servidor**: Tomada de la arquitectura del mismo nombre, esta característica responde a una separación de conceptos o preocupaciones. Separando los conceptos de la interfaz de usuario de aquellos relativos al almacenamiento de datos se mejora la portabilidad de la interfaz de usuario a múltiples plataformas y se mejora la escalabilidad simplificando los componentes del servidor.
- **Sin estado**: La comunicación debe ser sin estado, de forma que cada petición del cliente al servidor deba contener toda la información necesaria para ser entendida y procesada, sin poder tomar ventaja de cualquier contexto almacenado en el servidor. Así pues la clave de sesión es por tanto almacenada en el cliente. Esta restricción introduce las propiedades de visibilidad, confiabilidad y escalabilidad.
- **Almacén temporal (*cache*)**: Para mejorar la eficiencia de la red se pueden emplear almacenes temporales que guarden las respuestas dadas por los servidores a las peticiones de los clientes. Para evitar que éstos manejen información desactualizada, las respuestas deben incluir una etiqueta que las defina como almacenables o no.
- **Interfaz uniforme**: La interfaz debe ser uniforme de manera que las implementaciones de los servicios estén desacopladas permitiendo la evolución independiente de las partes.
- **Sistema de capas**: La arquitectura REST debe admitir el uso de múltiples capas de forma que cada componente no sepa que otras capas hay más allá de aquellas con las que interactúa directamente. En el caso de la filosofía cliente – servidor esto se puede reflejar en la forma en la que el cliente no tiene porqué saber si está tratando directamente con el servidor que le presta el servicio o con un intermediario.
- **Código bajo demanda**: REST permite a los clientes la funcionalidad de ser extendidos mediante la descarga y ejecución de código adicional. De esta forma se simplifican los clientes reduciendo el número de características requeridas para su implementación.

2.6. OSGi

Desde el concepto original de una Pasarela de Servicios Abierta (Open Services Gateway initiative, OSGi), la tecnología OSGi ha evolucionado hacia conjunto de especificaciones desarrolladas por la Alianza OSGi en las que se define un sistema de componentes dinámicos para Java. Con estas especificaciones se reduce la complejidad del software mediante la provisión de una arquitectura modular para sistemas distribuidos a gran escala, así como para el desarrollo de pequeñas aplicaciones empotradas [57].

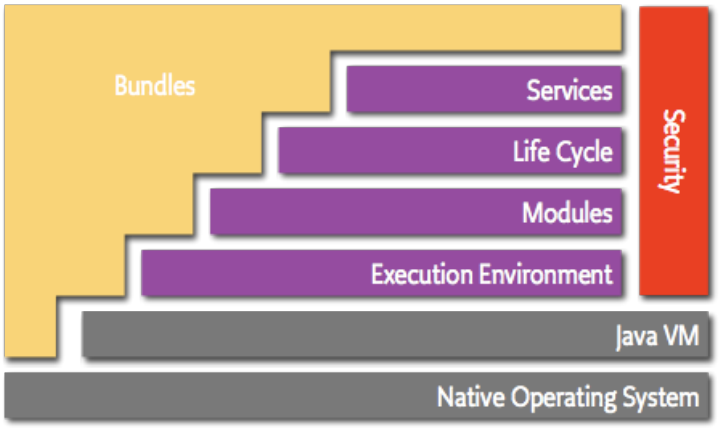

Figura 35. Arquitectura del modelo OSGi [57]

Un entorno OSGi facilita el desarrollo de aplicaciones simplificando su diseño en base a pequeñas estructuras modulares llamadas *bundles* o componentes, y que se comunican entre sí mediante el uso de servicios. De la descripción de la arquitectura ilustrada en la Figura 35 esta es la única parte que deben proporcionar los desarrolladores.

La capa de servicios descrita en la arquitectura conecta los componentes de forma dinámica mediante el uso de un modelo basado en los principios de publicación, búsqueda y asociación.

La idea general que se persigue es la de un sistema donde los componentes se pueden desplegar en tiempo de ejecución, empleando los métodos facilitados por la capa de ciclo de vida para instalarse, iniciarse, pararse, actualizarse y desinstalarse.

Una vez que un componente ha sido instalado e iniciado puede registrar uno o varios servicios en la plataforma, puede asociarse a los servicios proporcionados por otros componentes y puede permanecer a la escucha de ellos para ser notificado cuando aparezcan y desaparezcan.

2.7. Arquitecturas Orientadas a Servicio (SOA)

Una arquitectura orientada a servicio (Service Oriented Architecture, SOA) promueve la idea de ensamblar los componentes de una aplicación en una red de servicios ligeramente asociados para crear procesos flexibles y dinámicos de negocio así como aplicaciones ágiles que abarquen tanto a las organizaciones como a las plataformas de computación [58].

El modelo de referencia para una arquitectura de software orientada a servicios propuesto por OASIS [28] se centra en el concepto de propiedad, definiendo un SOA como «*un paradigma para la organización y utilización de capacidades distribuidas que pueden estar bajo el control de diferentes dominios de propiedad*». En este modelo se definen los conceptos principales de Visibilidad, Descripción del Servicio, Contexto de Ejecución, Efecto en el Mundo Real, Interacción, Contrato y Política.

Thomas Erl ha definido ocho principios específicos que debe cumplir un sistema orientado a servicios [59]:

- **Contrato normalizado de servicio**. Los servicios se adhieren a un acuerdo de comunicaciones, definido colectivamente por uno o más documentos de descripción del servicio.
- **Acoplamiento flexible de servicios**. Los servicios mantienen una relación que minimiza las dependencias y que sólo requiere que se mantenga el conocimiento de su existencia entre ellos.
- **Abstracción de servicio**. Los servicios ocultan su lógica interna, su implementación, del mundo exterior, exponiendo únicamente las descripciones publicadas en el contrato de servicio.
- **Reutilización de servicios**. La lógica de una aplicación se divide en servicios básicos que promueven su reutilización.
- **Autonomía de servicios**. Los servicios tienen el control sobre la lógica que implementan.
- **Servicios sin estado**. Los servicios reducen el consumo de recursos mediante la delegación de la gestión de la información de estado cuando es necesario.
- **Descubrimiento de servicios**. Los servicios se publican con información adicional que permita que puedan ser descubiertos e interpretados.

- **Composición de servicios**. Los servicios pueden participar activamente en la creación de otros servicios complejos, con independencia del tamaño y la complejidad de la composición.

Papazoglou además expone en [58] que los servicios pueden ser divididos en tres niveles de funciones: básico (nivel inferior), composición (nivel intermedio) y gestión (nivel superior).

2.8. Enterprise Service Bus (ESB)

2.8.1. Definición

La primera definición de un bus de servicios de empresa (Enterprise Service Bus, ESB) se debe a Roy W. Schulte, en su informe de predicciones para el año 2004 para Gartner [60]:

> Los buses de servicios de empresa (ESB) son un nuevo tipo de *middleware* que combina las características de varios tipos previos de *middleware* en un único paquete. Se apoyan en el uso de servicios Web mediante la implementación de SOAP y el uso de WSDL y UDDI. Muchos ESB soportan además otros estilos de comunicación que incluyen la garantía de entrega, y el modelo de publicación y suscripción. Y aquellos que no lo hacen lo harán pronto. Todos los ESB proporcionan algunos servicios de valor añadido además de los que se encuentran en cualquier *middleware* de comunicación básico, tales como la validación de mensajes, la transformación, el encaminamiento basado en el contenido, la seguridad, el descubrimiento de servicios en una arquitectura SOA, el balanceo de carga, la tolerancia a fallos y el registro de información. Algunos servicios forman parte del núcleo del EBS, mientras que otros se ejecutan como módulos adicionales. Además tienen una arquitectura distribuida en la que los servicios se ejecutan cerca de las aplicaciones, en lugar de en un servidor central. Y emplean XML como formato de mensaje, además de incluir con frecuencia la opción de usar otros formatos.

Cuadro 4. Definición de original de ESB por Roy W. Schulte para Gartner

Ante la falta de normalización en cuanto al término, hoy se concibe un ESB como un modelo de arquitectura de software usado para diseñar e implementar la interacción y la comunicación entre aplicaciones que interactúan mutuamente siguiendo los principios de SOA.

La idea clave de esta propuesta tecnológica es el uso del concepto de bus en una forma similar a la que se emplea en las arquitecturas de hardware. Según este principio, diferentes componentes de diferentes fabricantes empleando un sistema de mensajería común pueden funcionar sobre el mismo bus, dando lugar a una plataforma heterogénea de servicios interoperables.

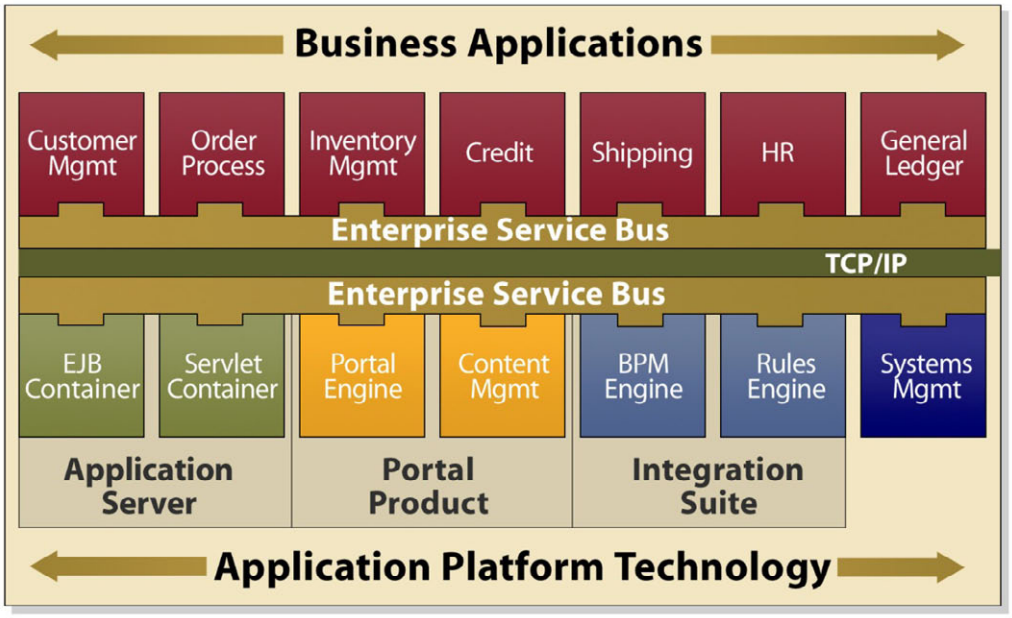

Figura 36. Visión ESB propuesta en el informe de Gartner [60]

En su libro titulado *Enterprise Service Bus* David Chappell describe un ESB como «*una plataforma de integración basadas en estándares que combina mensajería, servicios web, transformación de datos, y encaminamiento inteligente para conectar y coordinar con garantías la interacción de un significativo número de aplicaciones diversas implicando a empresas extendidas y manteniendo la integridad en las transacciones*» [61].

Con «*empresas extendidas*» se hace referencia a una organización a sus socios de negocio, que pueden estar separadas tanto en límites de negocio como físicos.

Un ESB se define típicamente por la lista de servicios que proporciona, siendo habituales:

- Mediación de transporte.
- Transformación dinámica de mensajes.
- Encaminamiento inteligente.
- Seguridad.

De esta forma un ESB simplifica la complejidad de la integración proporcionando una única infraestructura normalizada en la que se pueden desplegar y conectar las aplicaciones. Una vez conectadas en el ESB las aplicaciones o servicios pueden acceder a toda la infraestructura de servicios facilitada por el bus, incluyendo el resto de aplicaciones conectadas.

Teniendo en cuenta que no existe una definición normalizada que describa los servicios y componentes mínimos de un ESB, se hace difícil comparar y diferenciar entre las diferentes implementaciones existentes en el mercado. Por eso las diferencias suelen evaluarse atendiendo a las siguientes medidas:

- Soporte de despliegue/entornos de ejecución. Idealmente una solución ESB debería tener requisitos de despliegue flexibles para que pudiera ser distribuida a través de la empresa con facilidad.
- Modelo de contenedores y componentes. Idealmente una solución ESB debería emplear un modelo de contenedores normalizado, maximizando la compatibilidad y reduciendo la curva de aprendizaje necesaria para su adopción.
- Acoplamiento a otros componentes de la infraestructura. Idealmente una solución ESB debería proporcionar un acoplamiento flexible entre el ESB y los componentes adicionales, o al menos proporcionar una interfaz normalizada entre ellos. De esta forma el ESB puede ser extendido con facilidad.
- Dependencias. Idealmente una solución ESB debería depender únicamente de bibliotecas normalizadas de amplia aceptación y uso.

2.8.2. Fuse ESB Enterprise

Fuse ESB Enterprise es una solución ESB libre basada en Apache Service Mix, y que reduce la complejidad y elimina la dependencia del fabricante por estar basado en estándares y construido a base de tecnología de software abierto [62]. Sus características principales son:

- Es ligero y puede funcionar en la mayoría de las plataformas disponibles.
- Usa un marco OSGi para simplificar la creación de aplicaciones basadas en componentes.
- Proporciona un mecanismo que automatiza la creación y mantenimiento de los componentes OSGi.

- Cumple la especificación de Integración de Negocio en Java (Java Business Integration, JBI) [63].
- Puede asociarse a otras infraestructuras de servicio sobre una amplia variedad de protocolos de transporte y formatos de datos.
- Utiliza estándares y normativas en todo lo posible para limitar las dependencias.
- Acepta arquitecturas basadas en eventos.

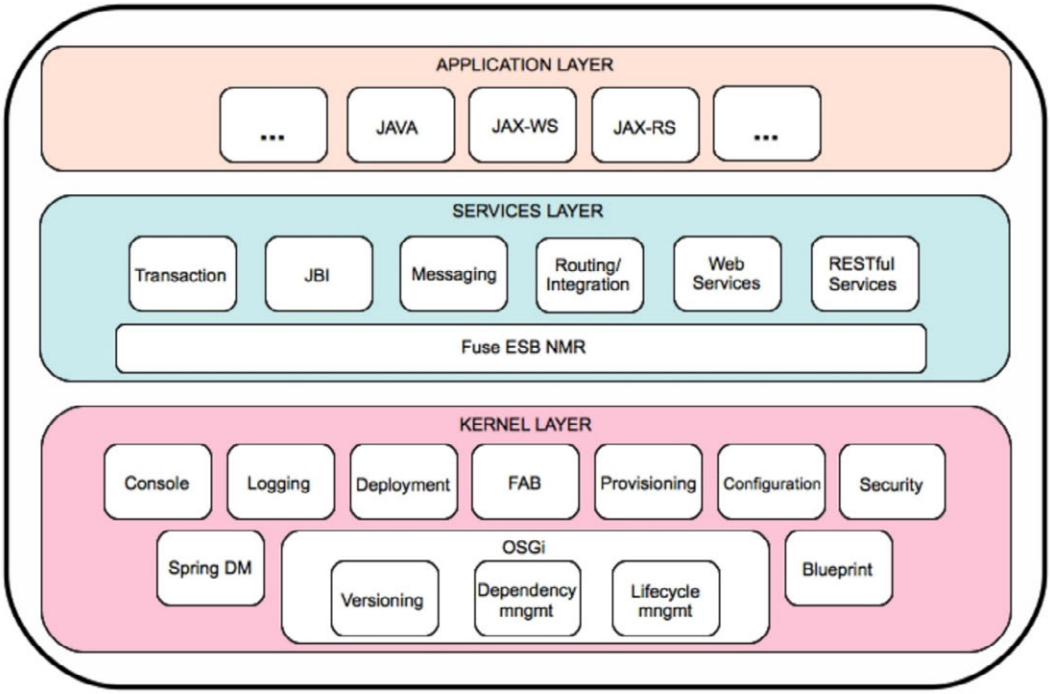

Figura 37. Niveles de la arquitectura empleada por Fuse ESB Enterprise [62]

La arquitectura interna de Fuse ESB se divide en tres niveles:

- En el nivel inferior el **nivel del núcleo** basado en Apache Karaf, un motor de ejecución basado en OSGi que proporciona un contenedor ligero en el que se pueden desplegar varios componentes y aplicaciones. Este nivel interactúa con el **nivel de servicios** para configurar, coordinar, y gestionar el registro y la seguridad, así como para manejar las transacciones por servicio.
- En el nivel intermedio el **nivel de servicios** que consiste en todas las interfaces e implementación de clases para cada uno de los servicios integrados. Interactúa con el **nivel de aplicación** para comunicarse con las aplicaciones desarrolladas por el usuario que quieren acceder a y utilizar estos servicios.

- En el nivel superior el **nivel de aplicación** que es donde residen las aplicaciones de usuario. Fuse ESB Enterprise proporciona múltiples interfaces de programación de aplicaciones con las que se pueden crear aplicaciones de tipo cliente y servidor que accedan y usen los servicios internos de la plataforma.

Las funciones que incluye Fuse ESB Enterprise se agrupan en componentes funcionales tal y como se ilustra en la Figura 38.

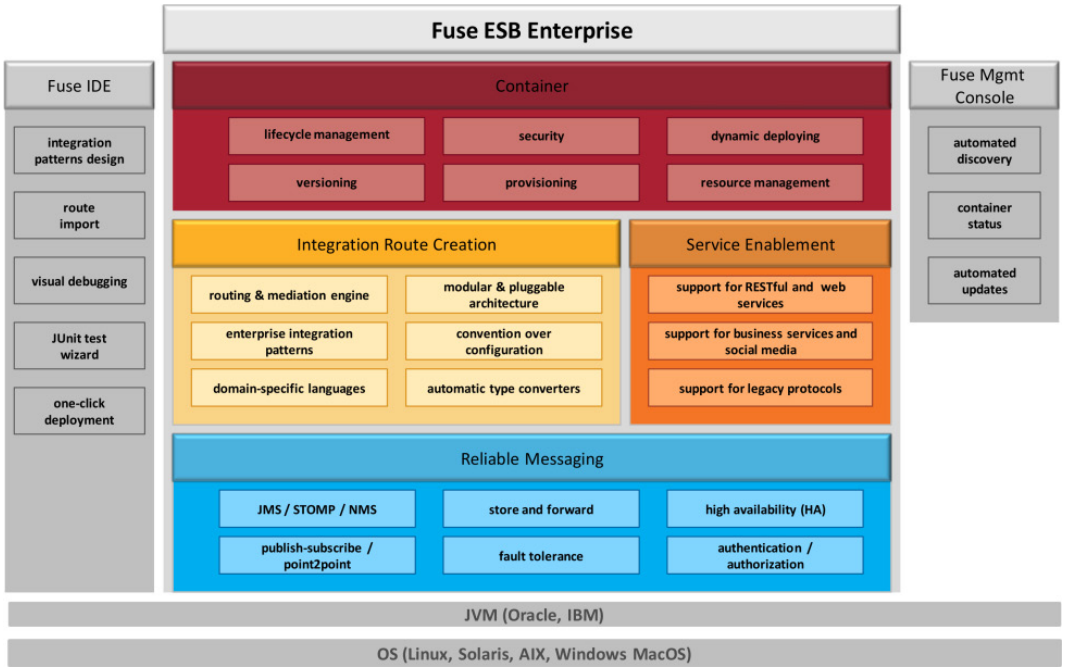

Figura 38. Modelo de componentes de Fuse ESB Enterprise [64]

Durante el desarrollo de este PFC Red Hat ha reorganizado su cartera de productos, y ahora Fuse ESB Enterprise forma parte de su suite jBoss siendo rebautizado como Red Hat jBoss Fuse, manteniendo la misma funcionalidad y diseño que descritos.

2.9. Composición de servicios

La composición de servicios implica la capacidad de combinar y coordinar múltiples servicios en un único servicio compuesto con el propósito de ofrecer una nueva funcionalidad no alcanzable mediante los servicios existentes. Los nuevos servicios así creados pasan a ser ofrecidos como servicios básicos que pueden ser finales, o formar parte en la composición de nuevos servicios [58].

El proceso de composición puede llevarse a cabo de forma manual o automática.

Habitualmente cuando las condiciones de composición son conocidas en el momento de diseñar la plataforma de servicios, ésta suele implementarse mediante una solución estática. Puede ocurrir también que las condiciones y necesidades para la composición de un servicio se den en tiempo de ejecución, es decir, con el sistema en funcionamiento. En estos casos se requiere de un esquema dinámico que permita realizarla, lo que otorga al sistema de una mayor flexibilidad [65, p. 105].

2.9.1. Modelos de composición

2.9.1.1. Orquestación

La orquestación de servicios es la descripción de la interacción a nivel de mensajes que ocurre entre los servicios que participan en la composición. De una forma más clara, un servicio orquestado puede considerarse como una composición proactiva, en la que existe una entidad responsable de ejecutar el flujo de acciones que resultan en el servicio compuesto.

El flujo de acciones que determina la forma en la que debe orquestarse el servicio compuesto de esta forma se suele describir mediante el uso de lenguajes de flujo que son interpretados y ejecutados por un motor de orquestación.

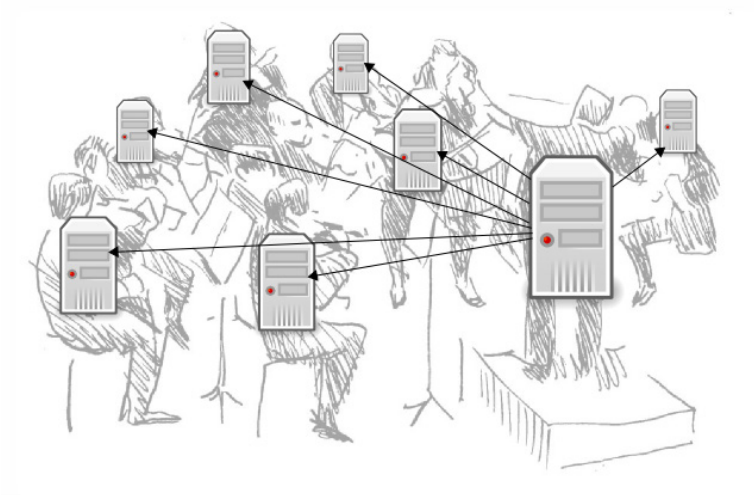

Figura 39. Representación de la orquestación de servicios [66]

2.9.1.2. Coreografía

La coreografía de servicios está típicamente asociada con el intercambio público de mensajes, reglas de interacción y acuerdos que ocurren entre múltiples puntos finales de proceso [58]. Al contrario que en la orquestación, en una coreografía de servicios no existe una entidad central que coordine y ejecute el flujo de acciones conducentes a la composición del servicio. En su

lugar todas las partes implicadas cooperan entre sí de acuerdo a las reglas de interacción definidas.

Al igual que con la orquestación, la coreografía es definida con lenguajes de composición creados al efecto. La diferencia radica en que mientras que la primera, como ya hemos visto, es ejecutada por un orquestador, en la segunda son las partes implicadas las que ejecutan la coreografía.

Algunas ventajas de la coreografía frente a la orquestación son una mejor escalabilidad, una mejor adaptación a la heterogeneidad, una mayor movilidad, y una mejor capacidad de alerta y adaptabilidad.

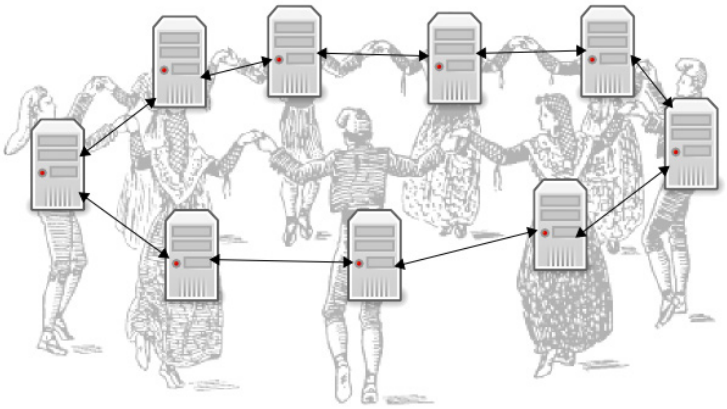

Figura 40. Representación de la coreografía de servicios [66]

2.9.2. Mecanismos de composición

2.9.2.1. Business Process for Model and Notation (BPMN)

La Notación para el Modelado de Procesos de Negocio (BPMN) es una notación gráfica normalizada por OMG que describe los pasos en un proceso de negocio. Esta notación ha sido diseñada específicamente para coordinar la secuencia de procesos y mensajes que fluyen entre los diferentes procesos participantes en un conjunto relacionado de actividades [67].

El objetivo de BPMN es facilitar la gestión de procesos de negocio, tanto para los usuarios técnicos como para los empresariales, proporcionando una notación que resulta intuitiva a la par que es capaz de representar semánticas complejas de procesos.

Un modelo BPMN está compuesto por diagramas simples que son construidos mediante la combinación de un conjunto limitado de elementos gráficos. Estos elementos pueden ser de cuatro

tipos diferentes: objetos de flujo, objetos de conexión, pistas, y artefactos.

Los objetos de flujo son los elementos principales, y se pueden dividir en tres grupos:

- **Eventos.** Se representan con un círculo y representan cosas que ocurren. Se pueden capturar o lanzar, y los hay de tres tipos:
  - **Eventos de inicio**.
  - **Eventos intermedios**.
  - **Eventos finales**.
- **Actividades.** Describen un tipo de trabajo que debe realizarse. Se distinguen cuatro tipos de actividades:
  - **Tareas**.
  - **Subprocesos**.
  - **Transacciones**.
  - **Llamadas a actividad**.
- **Pasarelas.** Son los puntos en los que la ruta de proceso puede dividirse o unirse, según las condiciones que se expresen. Se definen las siguientes:
  - **Exclusivas.**
  - **Basadas en eventos.**
  - **Paralelas.**
  - **Inclusivas.**
  - **Exclusivas basadas en eventos.**
  - **Complejas.**
  - **Paralelas basadas en eventos.**

Los objetos de flujo se relacionan entre sí mediante los objetos de conexión, que pueden ser de secuencia, de mensaje o de asociación.

Las pistas son mecanismos visuales para organizar y categorizar las actividades, y pueden ser de tipo piscina, que representa a los principales participantes en un proceso, o de tipo pista, que son aquellas que definen un rol o una función y de las que se componen las piscinas.

Finalmente los artefactos permiten a los desarrolladores añadir información al diagrama para hacerlo más legible y comprensible.

BPMN no es en sí mismo un lenguaje de composición de servicios, pero puede usarse para

describir servicios compuestos de tipo orquestado, e incluso contiene formas específicas para los servicios de tipo coreografiado. En la Figura 41 se muestra un ejemplo de una forma simple en la que se podría describir un servicio orquestado. Este diagrama se compone de tres pistas, siendo la superior la asociada a un orquestador que invoca mediante mensajes la actividad a otras dos entidades representadas por sendas pistas, y utiliza sus respuestas para generar la correspondiente al servicio orquestado.

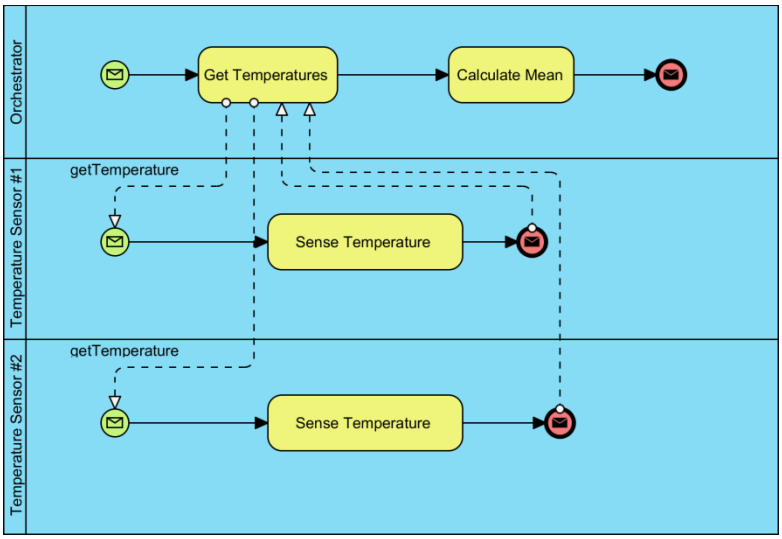

Figura 41. Ejemplo de orquestación en BPMN

2.9.2.2. Business Process Execution Language (WS-BPEL)

El Lenguaje de Ejecución de Procesos de Negocio (BPEL) es un lenguaje normalizado por OASIS que define un modelo y una gramática en XML para la descripción del comportamiento de un proceso de negocio basada en las interacciones entre los procesos y sus asociados [68]. La interacción con cada participante se realiza mediante interfaces de servicio web, y la estructura de la relación a nivel de interfaz se encapsula en un elemento que se llama *partnerLink*. El proceso BPEL define la manera en la que se coordinan las múltiples interacciones de servicio con los servicios asociados para conseguir un objetivo. Por tanto BPEL es un lenguaje de orquestación, ya que se mantiene un control centralizado de la ejecución desde el motor correspondiente que interpreta y ejecuta las instrucciones descritas.

WS-BPEL emplea múltiples especificaciones XML: WSDL1.1, XML Schema 1.0, XPath 1.0 y XSLT 1.0. Los mensajes de WSDL y las definiciones de tipos de XML Schema proporcionan el modelo de datos para los procesos WS-BPEL. XPath y XSLT proporcionan el apoyo para la manipulación de

datos. Todos los recursos externos y los asociados se representan como servicios WSDL. Además WS-BPEL es extensible para acomodarse a futuras versiones de estas recomendaciones.

Un proceso WS-BPEL es una definición reutilizable que puede ser desplegada de formas diferentes en múltiples escenarios, manteniendo un comportamiento de aplicación uniforme.

En el Cuadro 5 se incluye un ejemplo de la descripción de un proceso WS-BPEL.

```
<process name="HelloWorld" targetNamespace="http://jbpm.org/examples/hello"
         xmlns:tns="http://jbpm.org/examples/hello"
         xmlns:bpel="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
         xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/">

  <partnerLinks>
    <!-- establishes the relationship with the caller agent -->
    <partnerLink name="caller" partnerLinkType="tns:Greeter-Caller" myRole="Greeter" />
  </partnerLinks>

  <variables>
    <!-- holds the incoming message -->
    <variable name="request" messageType="tns:nameMessage" />
    <!-- holds the outgoing message -->
    <variable name="response" messageType="tns:greetingMessage" />
  </variables>

  <sequence name="MainSeq">

    <!-- receive the name of a person -->
    <receive name="ReceiveName" operation="sayHello" partnerLink="caller"
             portType="tns:Greeter" variable="request" createInstance="yes" />

    <!-- compose a greeting phrase -->
    <assign name="ComposeGreeting">
     <copy>
      <from expression="concat('Hello, ', bpel:getVariableData('request', 'name'), '!')" />
      <to variable="response" part="greeting" />
     </copy>
    </assign>

    <!-- send greeting back to caller -->
    <reply name="SendGreeting" operation="sayHello" partnerLink="caller"
           portType="tns:Greeter" variable="response" />

  </sequence>
</process>
```

Cuadro 5. Ejemplo de servicio compuesto con BPEL [69]

2.9.2.3. Choreography Description Language (WS-CDL)

El Lenguaje para la Descripción de Coreografías de Servicios Web (Web Services Choreography Description Language, WS-CDL) es un lenguaje basado en XML que describe la

colaboración entre pares mediante la definición de los comportamientos comunes y observables de cada participante en un proceso [70].

La especificación WS-CDL está dirigida a la composición de colaboraciones interoperables entre cualquier tipo de participante con independencia de la plataforma de soporte o el modelo de programación utilizado en la implementación del entorno de alojamiento. Una descripción de una coreografía es un contrato entre múltiples partes que describe la composición desde una perspectiva global, siendo WS-CDL el medio por el que se describe dicho contrato.

El uso de un contrato definido desde un punto de vista global aporta la ventaja de separar la lógica de ejecución interna de cada sistema individual participante en el proceso de colaboración de las secuencias que definen el proceso global. Esto a su vez implica que mientras estas secuencias globales permanezcan inalteradas, la lógica interna de los sistemas de cada participante puede cambiar, que la interoperabilidad seguirá estando garantizada.

WS-CDL define por tanto un lenguaje basado en XML para definir de forma declarativa dicho punto de vista global, cuyos objetivos son:

- **Reusabilidad.** La misma definición de la coreografía es utilizable por participantes diferentes que funcionan en otros contextos y con otro software.
- **Cooperación.** Las coreografías definen la secuencia de mensajes intercambiados entre dos o más participantes independientes describiendo como deben cooperar.
- **Colaboración múltiple.** Las coreografías deben poderse definir implicando cualquier número de participantes en el proceso.
- **Semántica.** Las coreografías pueden incluir documentación en formato legible para las personas, e información semántica para todos los componentes de la misma.
- **Componibilidad.** Las coreografías existentes pueden ser a su vez empleadas para definir nuevas coreografías y reutilizadas en contextos diferentes.
- **Modularidad.** Las coreografías pueden ser definidas empleando una facilidad de inclusión les permitan ser creadas de partes de otras coreografías diferentes.
- **Colaboración orientada a la información.** Las coreografías definen como los participantes progresan en una colaboración, a través del registro de la información intercambiada y de los cambios a la información observada, causando restricciones a cumplir y progresos a realizar.
- **Alineación de la información.** Las coreografías permiten a los participantes que

toman parte de ellas comunicarse y sincronizar su información observable.

- **Manejo de excepciones.** Las coreografías pueden definir como se manejan las situaciones excepcionales e inusuales que ocurren durante su actividad.
- **Transaccionalidad.** Los procesos que participan en una coreografía pueden trabajar de forma transaccional, con la capacidad de coordinar los resultados de larga duración, que incluyen múltiples participantes, cada uno con sus propias reglas y objetivos.

2.9.2.4. Orc

Orc es un lenguaje de programación concurrente no determinista creado por Jayadev Misra de la Universidad de Texas [71]. Proporciona un acceso uniforme a los servicios, incluyendo la comunicación distribuida y la manipulación de la información.

Empleando cuatro simples primitivas de concurrencia se puede emplear Orc para orquestar la invocación de sitios y así lograr un objetivo común mientras se gestionan los tiempos de espera, las prioridades, y los fallos [72].

Capítulo 3

Especificación nSOM

3.1. Introducción a la arquitectura nSOM

La arquitectura nSOM es una capa de intermediación, o *middleware*, diseñada para operar en entornos donde los recursos están limitados, inicialmente WSAN, y en general aplicable a las redes NTC que se describen en la arquitectura de referencia IoT-A. Por definición se define como una arquitectura orientada a servicios, y a ahí su nombre *nano Service Oriented Middleware*. Como capa de intermediación, nSOM es una capa de software que se sitúa entre las aplicaciones y el sistema operativo, facilitando la construcción de nuevas aplicaciones abstrayendo al programador de la heterogeneidad de redes, sistemas y componentes hardware [73]. El desarrollo de este middleware es un trabajo activo que se lleva a cabo por varios investigadores del Grupo de Redes y Servicios de Próxima Generación (GRyS) de la Escuela Universitaria de Ingeniería Técnica de Telecomunicación (EUITT) de la Universidad Politécnica de Madrid (UPM).

Las líneas de diseño principales de esta arquitectura siguen los paradigmas de la orientación a la actividad, la independencia de implementaciones y el enfoque a la composición, nociones que son propias de las propuestas de una arquitectura orientada a servicios (SOA). Siguiendo estos principios, el middleware proporciona un marco de trabajo o *framework* que permite la implementación de entidades de negocio basadas en servicios distribuidos de forma rápida, eficiente y de bajo coste [74]. A su vez ofrece una abstracción de los dispositivos que le permite funcionar con independencia de la plataforma y el hardware donde se implemente.

Operativamente nSOM hace uso de la tecnología de agentes como plataforma sobre la que desarrollar aplicaciones y componentes adicionales. Los agentes pueden acceder a los recursos de la arquitectura física donde se ejecutan, ya sea directamente mediante las funciones que les proporcione el sistema, o a través de la capa de abstracción empleando funciones nSOM. De igual forma interactúan con los componentes funcionales del middleware, y haciendo uso de sus servicios puede cooperar con otros agentes tanto locales como remotos, y siempre de forma abstracta para el propio agente.

La propuesta de la arquitectura nSOM se compone de tres niveles de abstracción, que quedan reflejados en la Figura 42. Estos tres niveles son en la parte inferior el nivel correspondiente al dispositivo físico. En el nivel intermedio se ubica la plataforma orientada a servicios, del que forman parte la capa de intermediación (middleware) nSOM y el marco de trabajo (framework)

nSOF. Finalmente en el nivel superior se encuentra la plataforma de composición y ejecución de servicios.

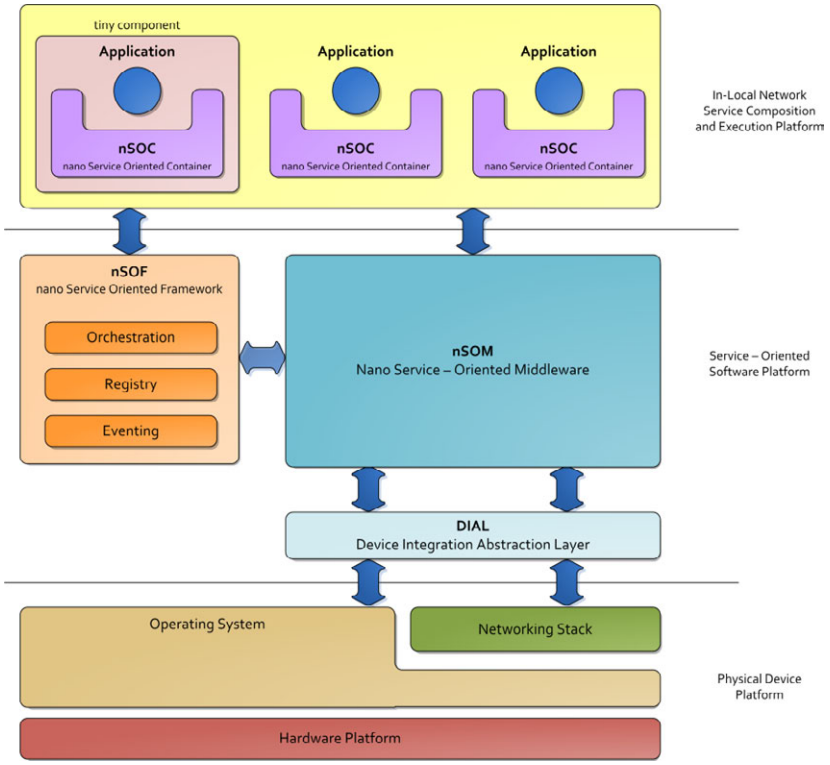

Figura 42. Arquitectura nSOM

El nivel de abstracción correspondiente al nivel físico agrupa las plataformas hardware de los nodos que forman parte de la red, el sistema operativo que utilicen y su soporte de red, incluyendo su pila de protocolos soportados. Representa así a las capacidades mínimas que debe poseer cualquier dispositivo físico sobre el que se desee desplegar y ejecutar la capa de intermediación, dando soporte así al resto de niveles de la abstracción.

El nivel de abstracción en el que se encuentra la plataforma de orientación a servicios es donde se ubican los componentes necesarios para abstraer al nivel superior de la heterogeneidad de redes y dispositivos que pueden darse en el nivel inferior. Se compone, como ya se ha descrito, del middleware nSOM, que ofrece las funcionalidades necesarias para la abstracción, y del framework asociado nSOF, que proporciona las entidades de apoyo necesarias para la composición y el registro de servicios.

En el último nivel de abstracción, el correspondiente a la plataforma de composición y ejecución de servicios, es donde se construyen las aplicaciones orientadas a servicios, empleando los componentes que proporciona la arquitectura.

3.1.1. Modelo de componentes del middleware nSOM

El middleware nSOM es el corazón de la arquitectura que se acaba de describir, y forma parte de su nivel intermedio como componente de abstracción. Como tal está integrado por servicios que se encargan de cumplir ese cometido, ofreciendo las funcionalidades necesarias.

Los servicios nSOM se agrupan conceptualmente en la forma en la que se muestra en la Figura 43 a continuación:

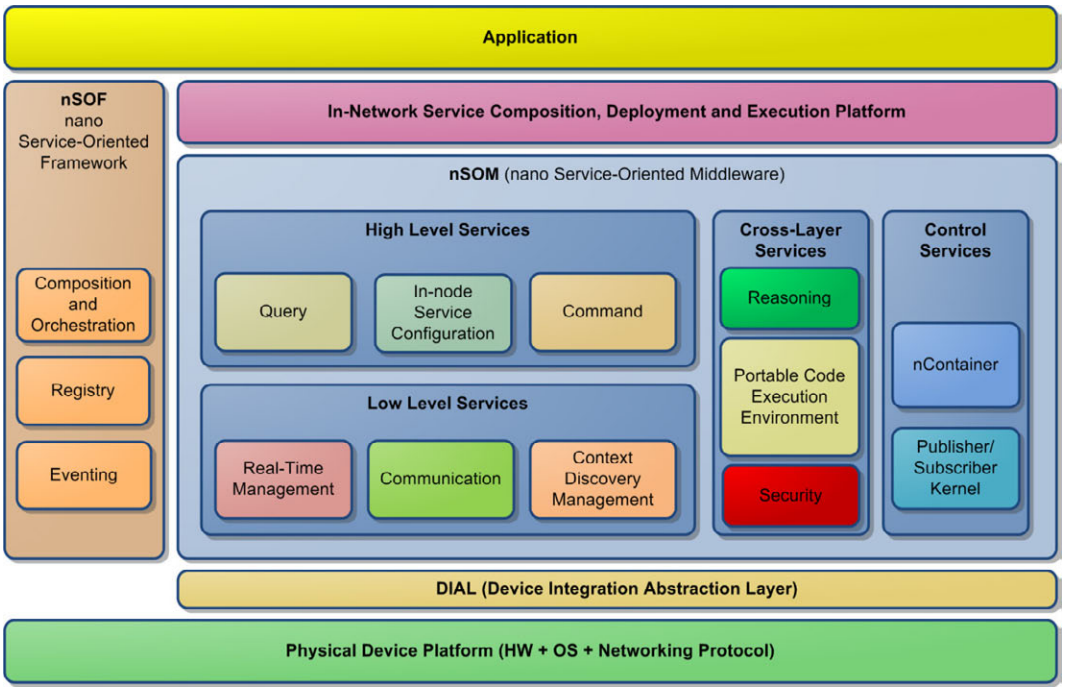

Figura 43. Aproximación del modelo de componentes y de los servicios en nSOM

Como puede apreciarse los servicios nSOM se encuentran inicialmente agrupados en cuatro grandes grupos: Servicios de Bajo Nivel (*Low Level Services*), Servicios de Alto Nivel (*High Level Services*), Servicios Multinivel (*Cross-Layer Services*) y Servicios de Control (*Control Services*).

El grupo de Servicios de Bajo Nivel contiene los servicios imprescindibles para el funcionamiento del middleware, y por extensión, del dispositivo donde residen. Son los empleados con mayor frecuencia e intensidad, y se agrupan de la siguiente forma:

- **Gestión en Tiempo Real** (*Real-Time Management*). En él se proporcionan los medios para la configuración de las capacidades de tiempo real del sistema operativo, siguiendo las necesidades de las aplicaciones desplegadas.
- **Comunicación** (*Communication*). Ofrece una abstracción de la capa de red mediante

una interfaz que proporciona todas las funciones necesarias para la transmisión de los mensajes, tales como el encapsulado y el encaminamiento.

- **Gestión de Descubrimiento del Contexto** (*Context Discovery Management*). Proporciona las capacidades de descubrimiento y gestión de servicios, que pueden emplearse en la capa de aplicación para localizar a los proveedores de servicios, así como para sincronizar la lógica de negocio entre los diferentes componentes desplegados en diferentes dispositivos.

En el grupo de Servicios de Alto Nivel se ubican los servicios accesibles por las aplicaciones desde una perspectiva global del sistema. Le proporcionan un valor añadido al nodo y a la red, logrando que el middleware sea adaptable a múltiples escenarios, dando soporte a la Plataforma de Composición, Despliegue y Ejecución de Servicios (*In-Network Services Composition, Deployment and Execution Platform*). Su agrupación es la siguiente:

- **Consulta** (*Query*). Proporciona al middleware la capacidad de administración de los mensajes de consulta, tanto para la comunicación interna al nodo entre componentes, como externa al mismo entre dispositivos.
- **Configuración de servicio interna al nodo** (*In-node Service Configuration*). Este grupo de funciones está orientado a proporcionar los medios para la configuración de los servicios del middleware, incluyendo por ejemplo, el ajusto del hardware del dispositivo, de las políticas de seguridad, el encaminamiento, las funciones de agregación, etc.
- **Mandato** (*Command*). Otorga al middleware la posibilidad de crear y añadir mandatos e instrucciones que permitan manejar y administrar las aplicaciones desplegadas.

Los servicios del grupo de Servicios Multinivel son accesibles desde todos los demás niveles. Se encargan de dar apoyo a los servicios de alto y bajo nivel, y son:

- **Entorno de Ejecución de Código Portable** (*Portable Code Execution Environment*). Proporciona los medios para el despliegue y la ejecución de código portable o portátil procedente de otros nodos o dispositivos.
- **Razonamiento** (*Reasoning*). Ofrece la capacidad de modificar dinámicamente el comportamiento de un nodo, así como de los servicios en él desplegados, siguiendo un conjunto de reglas interpretadas por un motor de inferencia. Es en cierta manera

el cerebro de la arquitectura, capaz de aportar una consciencia del contexto global al sistema siguiendo una filosofía distribuida.

- **Seguridad** (*Security*). Define los mecanismos de seguridad para ofrecer los servicios de confidencialidad, integridad y autenticación, tanto en el origen como en el destino.

Finalmente, el grupo de Servicios de Control proporciona la gestión del despliegue y ciclo de vida de los componentes de aplicación, así como la capacidad de comunicación entre componentes internos del middleware siguiendo una filosofía de eventos. Se compone de:

- **nContenedor** (*nContainer*). Realiza la administración del ciclo de vida de los componentes en el middleware, ofreciendo capacidades de programación y despacho. Mediante este servicio se realizan las tareas de instanciación, carga, inicialización, ejecución, parada y destrucción de los diferentes componentes.
- **Núcleo de eventos** (*Publish/Subscriber Kernel*). Proporciona las capacidades de publicación y suscripción a los componentes del middleware.

A los propios servicios del middleware se añaden los del framework, que dan soporte a la composición y registro de los servicios de aplicación.

3.1.2. Comparativa con la propuesta IoT-A.

Tomando de referencia la vista funcional de la arquitectura propuesta para IoT que se describe en 2.1.3.2.1, se pueden establecer una serie de relaciones entre los grupos y componentes funcionales de aquella, y los servicios del modelo de componentes de la arquitectura nSOM. Esta relación no es completa, en el sentido de que no todos los componentes nSOM tienen asociada una funcionalidad IoT-A, y viceversa. Esto es así en parte porque la propuesta IoT-A abarca a todo tipo de redes posibles en un sistema IoT, lo que incluye tanto a las redes sin restricciones NTU, como a las redes limitadas NTC del tipo que son las WSAN.

Así pues, es de esperar que sólo algunas partes fundamentales de la propuesta IoT-A correspondan con los componentes y servicios de nSOM. Y de igual manera, existen características específicas de las WSAN que son de interés intrínseca a las propias redes, y que no tienen correspondencia en IoT-A porque no están pensadas para interoperar con otro tipo de redes.

De una forma rápida se pueden observar en la Figura 44, una propuesta de proyección de los componentes y los grupos de servicios que se acaban de describir para nSOM sobre la vista funcional

propuesta por IoT-A que se describió en el segundo capítulo.

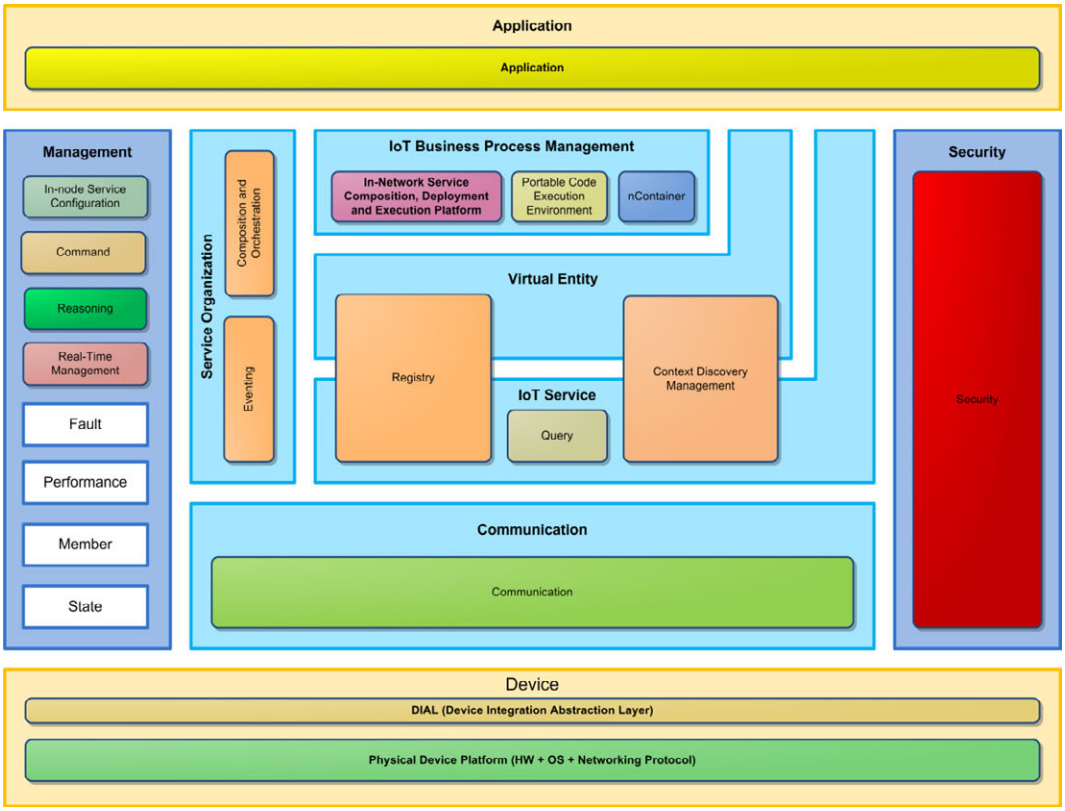

Figura 44. Propuesta de proyección de nSOM sobre IoT-A

Para describir esta propuesta de proyección, recordemos que la vista IoT-A se divide en siete grupos de funcionalidades principales. Tomándolos como referencia, los componentes de la arquitectura nSOM se pueden relacionar como sigue:

- **Grupo Funcional de Gestión de Procesos de Negocio IoT** (*IoT Business Process Management*). En este grupo se definían las funciones destinadas a la composición, despliegue y ejecución de los procesos de negocio implementados en las aplicaciones. Estas funciones son compatibles con los servicios desempeñados por los componentes de la Plataforma de Composición, Despliegue y Ejecución de Servicios, el Entorno de Ejecución de Código Portable y el nContenedor.
- **Grupo Funcional de Organización de Servicios** (*Service Organization*). Este grupo engloba las funciones de orquestación de servicios, incluyendo el procesado de eventos, y la composición. Es por tanto compatible con los componentes de Composición y Orquestación, y de Eventos, ambos parte del framework de la arquitectura nSOM.

- **Grupo Funcional de Entidad Virtual** (*Virtual Entity*). Entre otras funciones, en este grupo se definían las de registro y descubrimiento de las entidades virtuales, propias de los componentes Registro y Gestión del Descubrimiento del Contexto de la arquitectura nSOM.
- **Grupo Funcional de Servicio IoT** (*IoT Service*). Como con el anterior, se incluyen funciones de registro y descubrimiento, en este caso de servicios IoT, e igualmente asociadas a los componentes de Registro y Gestión del Descubrimiento del Contexto de nSOM. Además, el componente Consulta tiene cabida en este grupo funcional al ser el empleado para tramitar las peticiones de consulta de servicios en nSOM, tanto de forma interna al dispositivo como externa.
- **Grupo Funcional de Comunicaciones** (*Communication*). La equivalencia es directa con el componente de comunicaciones de nSOM. Cabe destacar este componente no incluye todas las funcionalidades descritas para el grupo equivalente en IoT-A, pero es asumible que pueda integrarlas en los escenarios donde se requieran.
- **Grupo Funcional de Seguridad** (*Security*). En IoT-A la propuesta incluye en este grupo las funciones de seguridad correspondientes a la autorización, intercambio y administración de claves, confianza y reputación, gestión de identidades y autenticación. El componente de seguridad de la arquitectura nSOM incluye confidencialidad, integridad y autenticación. Son, en definitiva los mismos principios, o derivados de los mismos, por lo que la equivalencia es directa entre un componente y otro.
- **Grupo Funcional de Administración** (*Management*). Este es el grupo donde existe menor equivalencia. En nSOM existen los componentes de Gestión de Tiempo Real, Configuración de Servicio interna al nodo, Mandato y Rendimiento, que pueden formar parte de este grupo, al estar relacionados con las tareas de configuración, rendimiento y operación del sistema, los servicios y los dispositivos. Quedarían por definir los componentes que asumieran las otras tareas de administración propuestas por IoT-A, como la administración de fallos (*Fault*), de miembros (*Members*) y de estado (*State*), así como parte de las tareas de rendimiento (*Performance*) no cubiertas por los componentes nSOM.

El único componente nSOM no asociado es el Núcleo de Eventos, que ofrece una funcionalidad propia para la interacción entre los componentes internos del middleware, y que no es por tanto directamente relacionable a las funcionalidades de servicios IoT.

3.2. Modelo de comunicaciones nSOM

nSOM proporciona un protocolo de descubrimiento de y consulta de servicios basado en el intercambio de mensajes. Los mensajes son Unidades de Datos de Protocolo (PDU, *Protocol Data Unit*) consistentes en una cabecera que contiene la Información de Control del Protocolo (PCI, *Protocol Control Information*), y la carga del mensaje (*Payload*), que está compuesta por un conjunto de objetos de longitud variable que aportan los datos necesarios en función del tipo de mensaje.

Los tipos de mensajes originalmente descritos en el protocolo y sus objetos asociados se pueden consultar en el trabajo de Jesús Rodríguez [75].

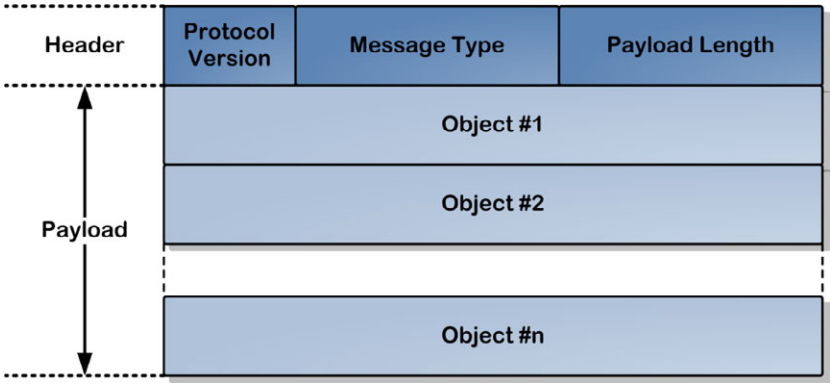

Figura 45. PDU genérica nSOM

Originalmente los objetos que forman parte de la carga de una PDU nSOM fueron concebidos para codificarse en binario, incluyendo información contextual asociada en la forma del empleo de un campo para discriminar entre tipos de objetos, y otro para la longitud de la carga del objeto, siendo esta una cadena de texto con la información necesaria.

En revisiones posteriores se ha optado por el uso de un objeto JSON, siguiendo un modelo basado en SMD y RDF que aportan un contexto semántico mediante una ontología reducida llamada nSOL (*nano Service Ontology Language*) [31]. En este trabajo se ha seguido la misma filosofía, por lo que los mensajes definidos para cumplir con el escenario que se propone en el siguiente capítulo incluirán un objeto JSON acorde a las definiciones que a continuación se van a tratar.

3.2.1. Descripción de los mensajes empleados en este PFC

De los tipos de mensaje definidos en [75] en este PFC se describen los siguientes: SERVICE SUBSCRIPTION, SERVICE AVAILABILITY, SERVICE SHUTDOWN y SERVICE UNSUBSCRIPTION.

3.2.1.1. Mensaje SERVICE SUBSCRIPTION

Sobre este mensaje se apoya la suscripción realizada por el componente de Composición y Orquestación de Servicios a los servicios que necesita para realizar su actividad de composición. El destinatario es un servicio remoto de Registro que a partir de la recepción del mensaje pasará a notificar de las altas de nuevos servicios acordes a las condiciones solicitadas.

Por definición este mensaje se transmite originalmente en multicast, a todos los servicios de registro conocidos, o broadcast, si se desconoce la presencia de servicios de registro en el sistema. Se deja abierta la posibilidad de que pueda emplearse una transmisión unicast en el caso de que el dispositivo sobre el que se ejecuta el componente de Composición y Orquestación de Servicios conozca de antemano la dirección del registro en el que quiere solicitar la suscripción.

El objeto JSON que compone el mensaje sigue el modelo definido en el Cuadro 6.

```
{
      “transport”: “j2me.radiogram”,
      “envelope”: “JSON-2.0”,
      “target”: “<Dirección>/<Agente de destino>”,
      “origin”: “<Dirección>/<Agente de origen>”,
      {
            “service”: “<Descripción del servicio>”
      }
}
```

Cuadro 6. Objeto JSON del mensaje nSOM SERVICE SUBSCRIPTION

El campo realmente importante con relación a este mensaje es el objeto anidado en el que se proporciona la descripción del servicio, entendiendo como tal el nombre del mismo, una definición abstracta, o incluso un comodín. De esta forma serían igualmente válidos como descriptores el uso de «Temperatura» para suscribirse a cualquier servicio de temperatura, «Temperatura en la habitación 123» para suscribirse a los servicios de temperatura ubicados en la habitación 123 o «*», para suscribirse a todos los servicios que se den de alta en el registro.

3.2.1.2. Mensaje SERVICE AVAILABILITY

Este mensaje se emplea para anunciar la existencia de un servicio del que previamente se ha solicitado la suscripción. El origen es el Registro que ha recibido las suscripciones previas, siendo el destino los componentes de Composición y Orquestación de Servicios que las han realizado.

Este mensaje se transmite por unicast, ya que el Registro conoce las direcciones de los suscriptores a los que enviar la notificación.

El objeto JSON de este mensaje sigue el modelo que se recoge en el Cuadro 7.

```
{
	“transport”: “j2me.radiogram”,
	“envelope”: “JSON-2.0”,
	“target”: “<Dirección>/<Agente de destino>”,
	“origin”: “<Dirección>/<Agente de origen>”,
	{
		“service”: “<Descripción del servicio>”,
		“provider”: “<Dirección>/<Agente>”,
		“operation”: “<Nombre de operación>”,
		“parameters”: [ <Parámetros> ]”
	}
}
```

Cuadro 7. Objeto JSON del mensaje nSOM SERVICE AVAILABILITY

El objeto es muy parecido al anterior. Se repite el campo con la descripción del servicio, donde el Registro emplea el mismo que fue usado en la suscripción para que el suscriptor sepa a qué solicitud se hace referencia. A esta información se le añade la correspondiente al proveedor del servicio, mediante su dirección e identificación del agente de servicio, el nombre de la operación con la que se invoca el servicio, y los parámetros necesarios, o el término «void» en caso de que no sean necesarios.

3.2.1.3. Mensaje SERVICE SHUTDOWN

Este mensaje lo envía el Registro a los suscriptores para informarles de que un servicio previamente notificado como disponible ha dejado de estarlo. El receptor esperado es el componente de Composición y Orquestación de Servicios, y la transmisión, como en el caso de la disponibilidad, se realiza en modo unicast.

El objeto JSON sigue el modelo que figura en el Cuadro 8.

```
{
	“transport”: “j2me.radiogram”,
	“envelope”: “JSON-2.0”,
	“target”: “<Dirección>/<Agente de destino>”,
	“origin”: “<Dirección>/<Agente de origen>”,
	{
		“service”: “<Descripción del servicio>”,
		“provider”: “<Dirección>/<Agente>”,
	}
}
```

Cuadro 8. Objeto JSON del mensaje nSOM SUBSCRIPTION SHUTDOWN

En este caso el receptor del mensaje necesita saber a qué servicio hace referencia la notificación y quién era su proveedor.

3.2.1.4. Mensaje SERVICE UNSUBSCRIPTION

Este mensaje lo envía el suscriptor a las notificaciones de registro, habitualmente el componente de Composición y Orquestación de Servicios, al Registro correspondiente donde se hayan formalizado las suscripciones. Su uso es para notificar al registro la baja de las suscripciones previamente realizadas, y el mensaje se puede transmitir por unicast o por multicast si son varios los registros en los que se cursa la baja de forma simultánea.

El objeto JSON para este mensaje es acorde al modelo del Cuadro 9.

```
{
      “transport”: “j2me.radiogram”,
      “envelope”: “JSON-2.0”,
      “target”: “<Dirección>/<Agente de destino>”,
      “origin”: “<Dirección>/<Agente de origen>”,
      {
            “service”: “<Descripción del servicio>”
      }
}
```

Cuadro 9. Objeto JSON del mensaje nSOM SERVICE UNSUBSCRIPTION

Como se puede apreciar el objeto es exactamente el mismo que para la suscripción, ya que la información necesaria para cancelarla es la misma que para darse de alta: la descripción del servicio.

3.2.2. Exposición y uso de servicios compuestos orquestados

3.2.2.1. Servicio orquestado orientado a consulta

Se entiende por servicio orquestado orientado a consulta a aquél servicio compuesto siguiendo el paradigma de la orquestación y que es accesible mediante consulta. Puede emplear para componer su servicio de entidad virtual tanto servicios de consulta, como servicios orientados a eventos. Por simplicidad se consideran tan sólo los primeros en esta descripción

El proceso de exposición y uso de un servicio orquestado orientado a consulta se puede dividir en tres subprocesos: composición y exposición, uso y cierre. El diagrama de secuencias que detalla el intercambio de mensajes para la composición y la exposición se muestra en la Figura 46.

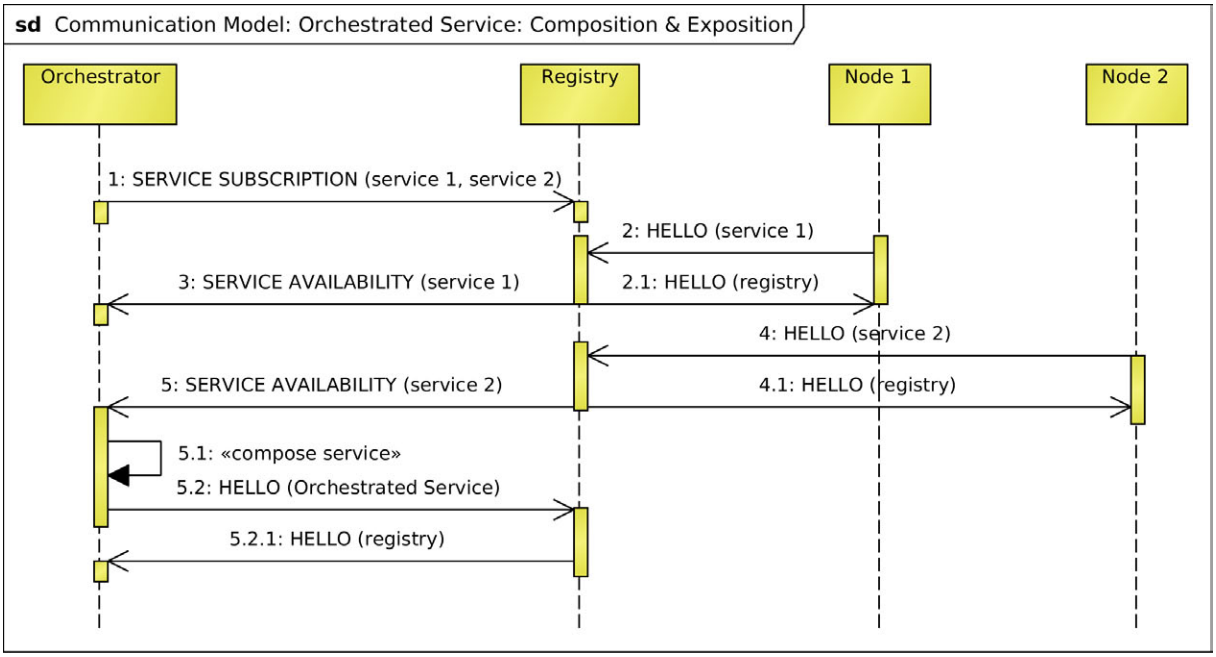

Figura 46. Composición y exposición de un servicio basado en servicios simples

Este caso de ejemplo se inicia con la petición de suscripción realizada por el orquestador al registro. En esta petición se le indica la lista de servicios, siguiendo los mensajes descritos en 3.2.1.

En este caso se parte del supuesto de que los servicios requeridos no se encuentran en ese momento dados de alta en el sistema. Así, el orquestador permanece inactivo, esperando del registro las notificaciones de disponibilidad de los servicios solicitados.

En (2) el nodo «Node 1» da de alta en el registro uno de los servicios requeridos por el orquestador. El registro entonces notifica la disponibilidad del mismo al orquestador en (3). Lo mismo sucede en (4) y (5) para el servicio expuesto por «Node 2».

Al recibir esta última confirmación, el orquestador ya cuenta con la información necesaria para componer el servicio. Ejecuta las instrucciones internas que sean necesarias según el caso, y lo da de alta en (5.2) en el registro como nuevo servicio orquestado.

Una vez que el nuevo servicio orquestado ha sido dado de alta en el registro, puede ser consultado por cualquier tipo de usuario, siguiendo la definición dada en el modelo de referencia IoT-A. La secuencia de mensajes se describe en la Figura 47.

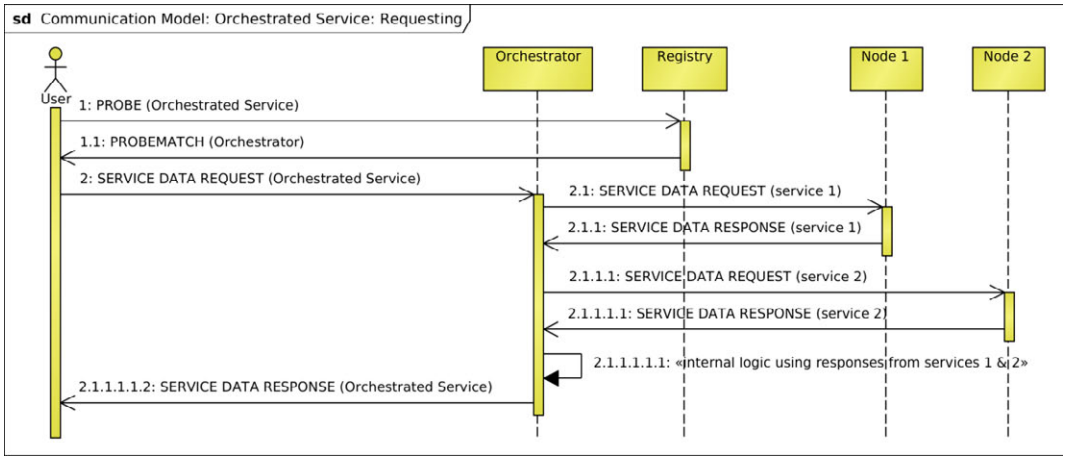

Figura 47. Consulta de un servicio orquestado basado en servicios simples

En este ejemplo el usuario primero consulta al registro por el servicio empleando el mensaje PROBE, y recibiendo el correspondiente PROBEMATCH conteniendo la información necesaria para acceder al servicio solicitado, que puede incluir la dirección del nodo que presta el servicio, y la información semántica necesaria para saber cómo acceder a él. Ambos mensajes están fuera del propósito de esta memoria, y se emplean en este diagrama sólo como prueba de concepto de su uso por parte de usuarios que formen parte de la misma WSAN.

Habitualmente el usuario se encontrará en una red NTU, y accederá al servicio mediante una pasarela, que siguiendo el modelo de referencia IoT-A podrá contener un registro y facilitar los medios para realizar la consulta directamente al prestador de servicios a través del servicio de reenvío de la pasarela.

Una vez que el usuario dispone de la información necesaria para invocar el servicio, envía la petición, esta vez ya directamente al orquestador (2). Al recibir la petición éste ejecuta la secuencia de instrucciones que definen la composición, que en éste caso requiere a su vez de la solicitud de servicios a otros nodos.

Finalmente, una vez tiene las respuestas de los servicios necesarios, realiza las operaciones internas necesarias con ellas generando su propia respuesta para el usuario.

Hay dos circunstancias en las que un orquestador puede dejar de prestar el servicio compuesto: por la baja repentina de alguno de los servicios necesarios en la orquestación y por iniciativa propia. La secuencia de mensajes cuando un nodo que forma parte de un servicio orquestado se muestra en la Figura 48.

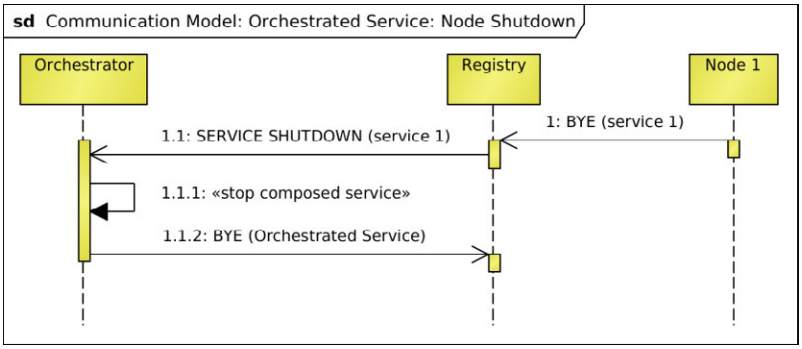

Figura 48. Cierre de un servicio simple necesario para la orquestación de un servicio

En este caso de forma ajena al orquestador uno de los nodos que presta uno de los servicios necesarios para la composición se da de baja en (1). El registro al recibir el mensaje de BYE correspondiente le notifica al orquestador que uno de los servicios a los que estaba suscrito ha dejado de estar disponible.

El orquestador entonces para cualquier lógica que pudiera estar ejecutando en relación al servicio compuesto y que necesitase el servicio que se acaba de dar de baja. Seguidamente notifica al registro la baja del servicio orquestado.

Eventualmente el orquestador puede dejar de prestar servicios en algún momento. Ante tal situación el cierre ordenado del servicio se realizaría tal y como se describe en la Figura 49.

En este caso el cierre se inicia anulando la suscripción en el registro a los servicios necesarios para la composición en (1) y (2), para seguidamente dar de baja el servicio compuesto en (3).

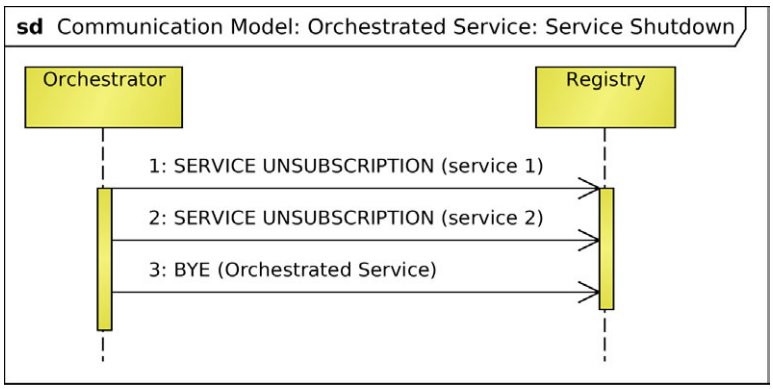

Figura 49. Cierre del servicio orquestado

3.2.2.2. Servicio orquestado orientado a eventos

Además de servicios básicos de consulta como los descritos en la sección anterior, se pueden

componer servicios orientados a eventos, basados tanto en el consumo de servicios simples como en el de servicios orientados a eventos.

En el ejemplo que se va a emplear para ilustrar esta situación el servicio orquestado va a ofrecer un servicio orientado a eventos basado en la monitorización del resultado de la consulta de dos servicios simples, y el consumo de los eventos de un tercer nodo. La secuencia de mensajes empleados en la composición y exposición del servicio se ilustra en la Figura 50. El modelo de comunicación de la parte de eventos está basado en el trabajo de María Jesús Martínez [76].

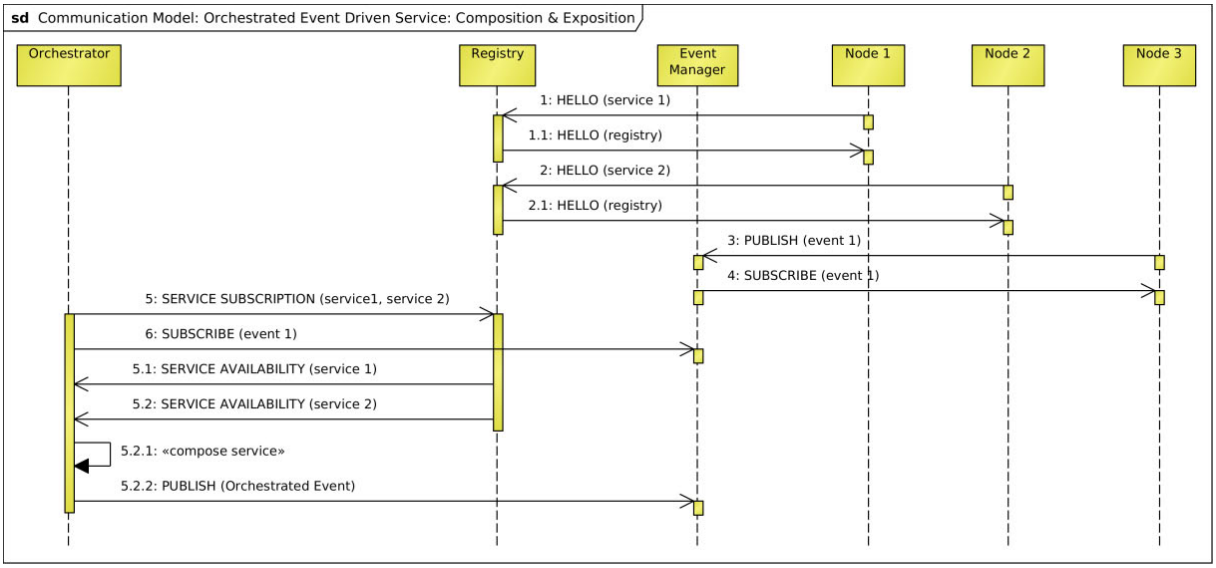

Figura 50. Composición y exposición de un servicio orquestado orientado a eventos

Para diferenciar y exponer una situación diferente a la mostrada en la Figura 46, en esta ocasión todos los servicios, tanto básicos como de eventos, ya se encuentran dados de alta cuando el orquestador se suscribe a ellos en los registros correspondientes. En esta ocasión por tanto los mensajes de disponibilidad que notifica el registro de servicios simples se envía de forma inmediata tras recibir la solicitud de suscripción. De la misma manera se suscribe en el gestor de eventos al evento que necesita para la composición.

Entonces, y al igual que en el caso de la orquestación de un servicio orientado a consulta, el orquestado ejecuta la lógica interna que necesite para componer el servicio, y lo expone dándose de alta esta vez en el gestor de eventos, como productor de un nuevo tipo de eventos compuestos.

A partir de ese momento cualquier usuario del sistema puede suscribirse al nuevo evento orquestado, que en el ejemplo puede generarse de las dos formas que se ilustran en la Figura 51.

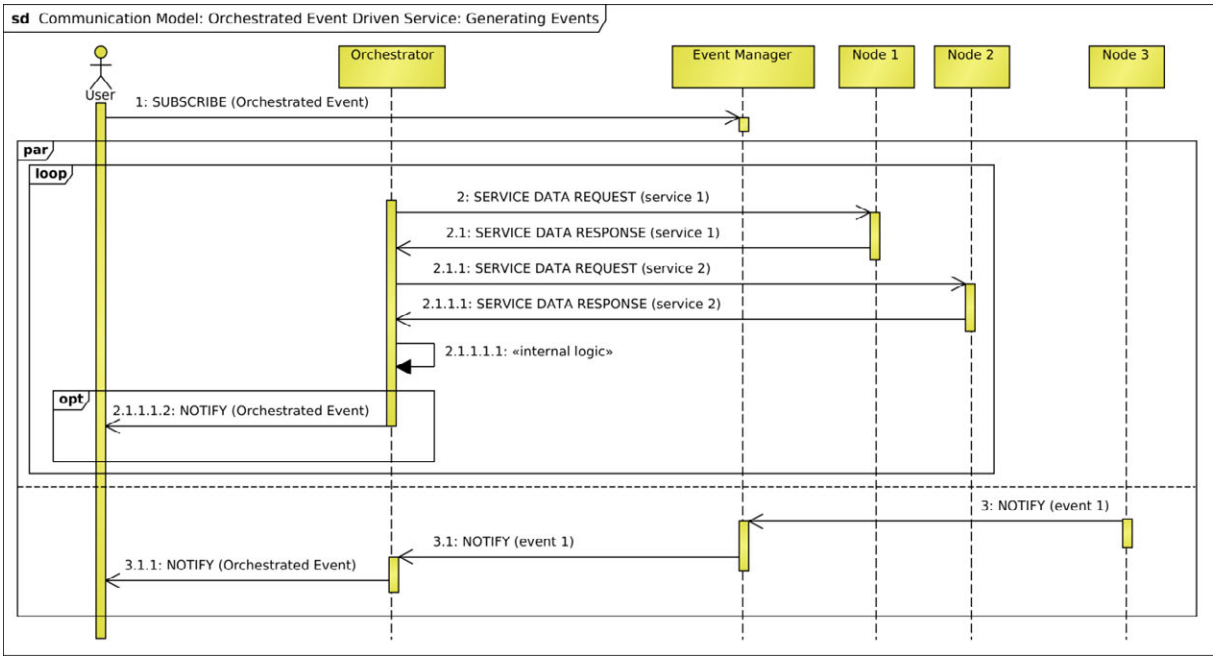

Figura 51. Generación de eventos mediante orquestación de servicios

En un primer paso un usuario se suscribe al evento del orquestador en el gestor de eventos. A partir de ese momento será notificado de un nuevo evento cuando se den las condiciones necesarias, que pueden ser de dos tipos.

De un lado el orquestador ejecuta periódicamente una monitorización de los servicios prestados por los dos nodos de servicio, y aplica procesa los resultados obtenidos. Si estos cumplen alguna condición definida en la orquestación, se genera el evento.

De igual forma, si se recibe un evento procedente del nodo generador de eventos, este se propaga como un nuevo evento orquestado.

En un caso real los nodos de servicio podrían ser nodos de temperatura, y el nodo de eventos un nodo que detectase una presencia en su proximidad y generase un evento en consecuencia. El orquestador podría ser entonces un nodo de alarmas que sirviera para centralizar todas las suscripciones a alarmas en un dispositivo con mayores recursos que los presenten en el resto del sistema. El orquestador, o nodo de alarmas en el ejemplo, monitorizaría entonces los nodos de temperatura, y generaría un evento de alarma si la temperatura medida superase unos valores de umbral definidos. De igual forma generaría un evento de alarma si el nodo de presencia notifica de la detección de una.

En cualquier caso el principio que se expone es genérico y abstracto, procurando ilustrar que

en todo momento se cumplen los principios SOA con los que ha sido concebida la arquitectura.

El cierre de un orquestador que presta un servicio orientado a eventos se realiza en las mismas formas que ya se han tratado para la composición anterior, por lo que los mismos diagramas siguen siendo de aplicación. Quedaría en todo caso la consideración de que alguno de los servicios que se diera de baja no implicase el cierre de toda la orquestación, sino de sólo una parte.

Por ejemplo, si se dejase de prestar el servicio ofrecido por el Nodo 1, el orquestador no tendría que dar de baja el servicio orquestado pues aún podría notificar de los eventos generados por el Nodo 3. Y de igual manera ocurriría en caso de que fuera el Nodo 3 el que dejase de prestar servicio, pues aún se podrían notificar los eventos que estuvieran asociados a la monitorización de los servicios de los Nodos 1 y 2.

3.3. Diseño de la pasarela nSOM sobre el ESB

3.3.1. Justificación de diseño

Tanto en la literatura acerca de los sistemas IoT, como por extensión en el ámbito del proyecto Web of Objects, uno de los problemas que se plantea es el de la interoperatividad con redes cuyos recursos son limitados. Infraestructuras reales que puede que no sean directamente accesibles para las aplicaciones principales de los sistemas en los que se integran. Son situaciones en las que los dispositivos no cuentan con los suficientes medios para atender largos procesos de comunicación, en situaciones en las que el acceso a sus recursos no puede ofrecerse siempre en tiempo real, y que incluso pueden no ser capaces de gestionar el posiblemente amplio número de solicitudes que puedan confluir en un momento dado.

En tales situaciones una solución adecuada es proporcionar el acceso a estos dispositivos y a sus servicios en la forma de objetos o entidades virtuales operados por otros dispositivos con mayores capacidades. Una virtualización de los servicios que suele llevarse a cabo mediante el empleo de pasarelas y capas de intermediación que se hacen cargo del procesado de los datos y de la gestión de las comunicaciones teniendo en cuenta las restricciones de los objetos reales.

Atendiendo a estos principios, se propone para la arquitectura nSOM una pasarela que cumpla con los principios descritos, y en particular adopte las funciones recomendadas en la propuesta de la arquitectura IoT. En particular, la pasarela que se ha desarrollado para este proyecto

ofrece la traducción e intermediación entre redes, sirviendo en la definición básica como un puente entre la WSAN y el resto de redes a las que se pueda conectar la propia pasarela, que en este caso se limita a Internet. A esta funcionalidad se le ha añadido la de registro de dispositivos y servicios, tal como propone IoT-A para la operación en redes pobladas por dispositivos de bajas prestaciones. Y todo ello además añadiendo una virtualización de los servicios ofrecidos por éstos mediante una interfaz de acceso REST que abstrae al usuario de la realidad de los dispositivos que prestan el servicio.

3.3.2. Diseño de la pasarela

En la Figura 52 se muestra el diagrama UML de clases desarrolladas para la puesta en marcha de la pasarela. La arquitectura incluye otras clases adicionales que no se describen al no ser objeto de estudio, si bien se hace referencia a otras dos clases de otros paquetes que son empleadas de forma directa por las utilizadas en la pasarela.

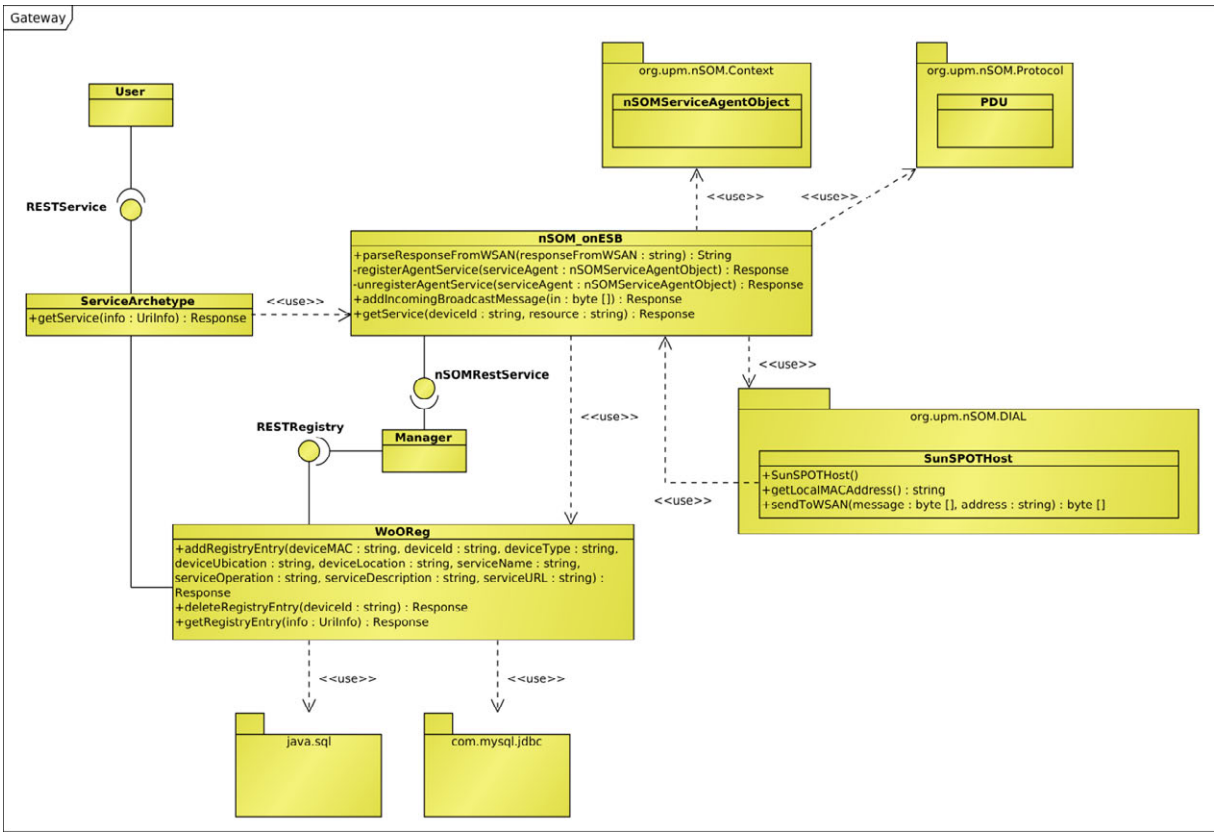

Figura 52. Diseño de clases de la pasarela

La clase nSOM_onESB es el corazón de la pasarela. En ella reciben y procesan todas las PDU nSOM recibidas desde la WSAN así como las peticiones realizadas por los usuarios. Para ello se apoya en la clase SunSPOTHOST del DIAL (*Device Integration Abstraction Layer*) para las comunicaciones

con la WSAN, y en los arquetipos de servicio, representados por la clase ServiceArchetype, para recibir las peticiones de los usuarios.

Así pues, la clase SunSPOTHOST es la responsable de la pasarela físicamente a la WSAN mediante un sumidero que permite el envío y recepción empleando una interfaz IEEE 802.15.4. Esta clase cuando recibe algún mensaje que deba ser procesado por la pasarela, se lo notifica a la clase nSOM_onESB mediante el método *addIncomingBroadcastMessage*. Esta clase a su vez lo procesa, y si procede, lo registra llamando empleando a su vez la clase WoOReg, que es una clase específica para el acceso al registro creado para el proyecto WoO, y que sigue una interfaz genérica para poder ser reutilizada en otros sistemas IoT cuando sea necesario.

Todo nuevo dispositivo dado de alta en el registro provoca la instanciación de un componente de servicio que hará las funciones de intermediario ofreciendo una interfaz REST para acceder a los servicios proporcionados. Estos nuevos componentes son representados en el diagrama por la clase *ServiceArchetype*.

De esta forma cuando un usuario, representado en el diagrama por la clase User, realiza una petición de servicio mediante la interfaz REST creada al efecto, ésta invoca al método correspondiente de la clase nSOM_onESB, quien a su vez consulta los datos necesarios en el registro para componer el mensaje y la PDU de petición de servicio, y emplear el método *sendToWSAN* de la clase SunSPOTHost para enviarla a la red, y obtener la respuesta, que también es procesada por nSOM_onESB para entregarla a la interfaz REST y mostrarla al usuario en un formato accesible.

Finalmente en el diagrama se muestran sendas interfaces REST de acceso a WoOReg y nSOM_onESB, que son empleados para tareas internas de administración, tales como la inserción manual de datos en el registro atendiendo a las necesidades de servicios remotos sin facilidades para el registro automático.

3.3.3. Diseño de la base de datos de dispositivos y servicios

Para la implementación del registro de dispositivos y servicios se ha recurrido al uso de una base de datos SQL definida según el modelo entidad-relación que se muestra en la Figura 53.

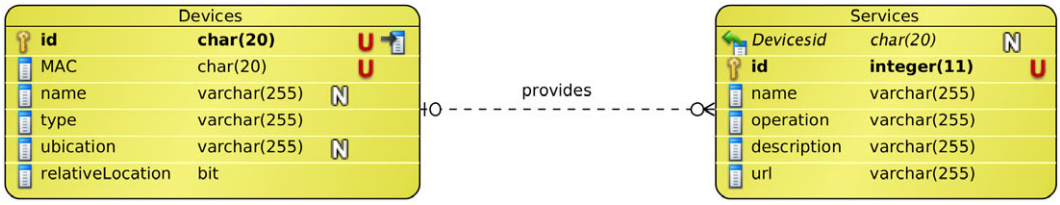

Figura 53. Diagrama Entidad-Relación del registro de dispositivos y servicios

Se trata de una base de datos relacional formada por dos entidades y una relación. De un lado tenemos la entidad «Devices», cuyos atributos se describen en la Tabla 32.

Atributo	Descripción
Id	Clave primaria basada en la identidad física del dispositivo.
MAC	La dirección MAC del dispositivo, que es única.
Name	El nombre del dispositivo, si lo tiene. Este atributo puede no ser declarado.
Type	El tipo de dispositivo, atendiendo a sus funcionalidades. En el escenario que va a emplearse para la validación sólo puede tomar los tipos «temperature» y «presence».
ubication	La ubicación del dispositivo, por ejemplo «Habitación 123».
relativeLocation	Este registro por definición puede registrar dispositivos y servicios remotos además de los ofrecidos por la WSAN a la que está conectado. Con este atributo se recoge la información sobre la ubicación relativa del dispositivo, que puede tomar los valores de «local» o «remote».

Tabla 32. Atributos de la entrada «device» en el Registro

En cuanto a la entrada «services», sus atributos se describen en la Tabla 33.

Atributo	Definición
Id	Clave primaria autogenerada.
Name	Nombre del servicio.
operation	Nombre de la operación con la que se invoca el servicio.
description	Descripción del servicio.
url	Punto de entrada REST al servicio.

Tabla 33. Atributos de la entrada «service» en el Registro.

3.4. Diseño del motor de orquestación nSOM

3.4.1. Justificación del diseño

El uso de herramientas de composición y orquestación en una arquitectura orientada a servicios introduce la posibilidad de crear nuevos servicios no concebidos originalmente en el diseño del sistema, y que tienen por ello la consideración de servicios virtuales, si bien desde la perspectiva de sus consumidores no existen diferencias con respecto a los servicios simples.

Habitualmente en las redes caracterizadas por una limitación de recursos se ha venido optando por soluciones específicas para cada escenario concreto, resultando en la necesidad de reprogramar el servicio orquestado para adaptarlo a cada dominio y escenario.

En este caso se plantea adoptar un modelo general con la intención de que pueda aplicarse en el futuro a diferentes escenarios con los mínimos cambios necesarios, empleando para ello los conceptos de los sistemas de administración de procesos de negocio que se han visto en el Capítulo 2, sección 2.9.

A tal fin se propone el diseño de un motor de orquestación capaz de interpretar unas reglas sencillas que le permitan componer nuevos servicios en base a otros existentes en la plataforma mediante el uso modelo de comunicaciones descrito en la sección 3.2.2.

3.4.2. Reglas de orquestación de servicios

Inicialmente dado el carácter de limitación de recursos sobre el que se pretende desplegar este tipo de orquestación, no resulta práctico el uso del lenguaje de procesado de negocios BPEL descrito en el Capítulo 2, sección 2.9.2.2. Por un lado se trata de un lenguaje basado en XML, con una gran recarga de elementos, por lo que su manejo en dispositivos que tengan limitados sus recursos puede ser impracticable. Por otro lado la definición de orquestaciones en BPEL resulta bastante compleja, si bien puede recurrirse a herramientas gráficas que abstraigan de las reglas finales de composición, pero que requieren igualmente de una formación previa para su manejo.

Otras soluciones como Orc, o BPELScript [77] intentan aligerar la descripción de la composición de servicios empleando sintaxis propias de los lenguajes de programación, si bien emplean un amplio conjunto de instrucciones que se alejan de las necesidades que pueden encontrarse en una red con recursos limitados.

En la propuesta de diseño que se realiza el motor de orquestación inicialmente aceptará un conjunto muy limitado de reglas básicas expresadas en JSON, centradas en el escenario de este proyecto. Se considera interesante explorar en un futuro la extensibilidad de este conjunto de reglas para poder dar lugar a orquestaciones m

3.4.3. Diseño del Orquestador

En la Figura 54 se muestra el diagrama UML de las clases planteadas en el diseño de la propuesta para el subsistema de orquestación. Como en el caso de la pasarela el diagrama incluye otras clases y paquetes adicionales que no se describen al no ser objeto de estudio, pero a las que se hace referencia por ser empleadas de forma directa por las clases propuestas.

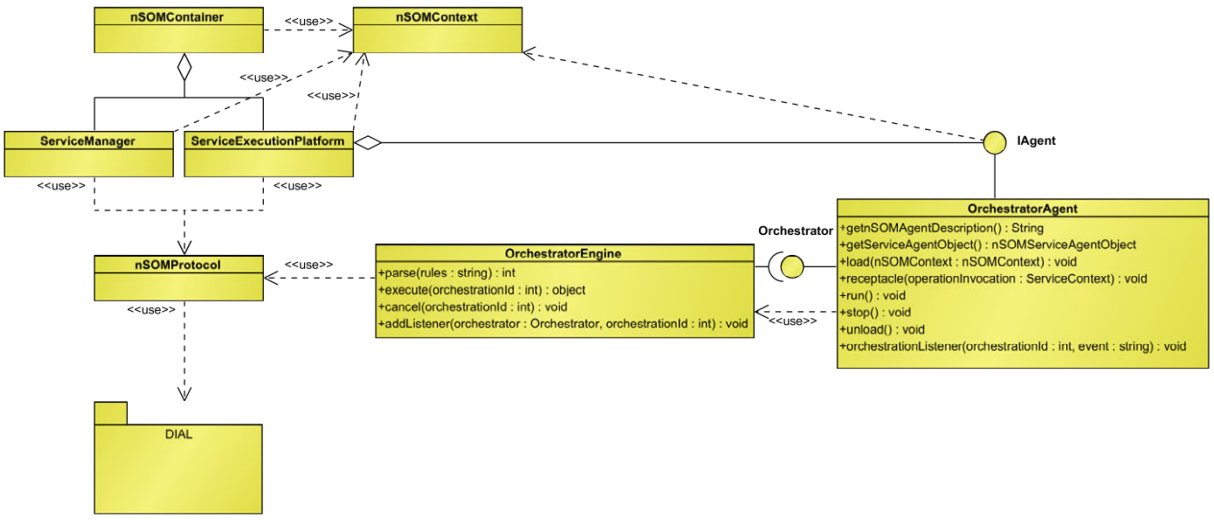

Figura 54. Diseño de clases de la propuesta de orquestación

En este caso el corazón del diseño se centra en la clase *OrchestratorEngine*. Esta clase es la responsable de recibir las reglas de composición de acuerdo a un patrón predefinido, e interpretarlas. El proceso de interpretación lleva consigo asociadas las tareas de asociar las reglas a una entrada en un registro interno para poder ser invocadas posteriormente, y la ejecución de las instrucciones necesarias en la plataforma nSOM para llevar a cabo la composición.

Esta clase utiliza la interfaz *Orchestrator* para notificar a quienes la implementen los eventos relativos a la orquestación solicitada mediante la invocación al método *orchestratorListener*. De esta forma el motor de orquestación puede notificar a quien lo haya invocado cuando una orquestación solicitada está disponible para su realización atendiendo al modelo de comunicaciones descrito con

anterioridad, o cuando ha ocurrido un evento que afecta a la orquestación, como por ejemplo el cierre de alguno de los servicios necesarios en la composición, o la recepción de un evento que forma parte de la propia orquestación y que en las reglas se especifica que debe ser atendido por quien las ha facilitado.

De otra parte tenemos la clase *OrchestratorAgente*, que es un agente de servicio más sobre la plataforma nSOM, con la particularidad de que su actividad se desarrolla a través de la composición de servicios. Implementa todas las clases propias de la interfaz de agentes, y es la responsable de invocar al método *parse* del motor de orquestación con las reglas que definen la composición solicitada. Esta invocación devuelve un identificador de orquestación, que será con el que se identificarán las siguientes acciones referidas a ella, como por ejemplo para cancelar una orquestación previamente solicitada, o para recibir las notificaciones de eventos asociadas a ella.

Capítulo 4

Validación del sistema y análisis de resultados

4.1. Introducción

En los capítulos anteriores se ha introducido la tecnología de referencia en el ámbito de desarrollo de este proyecto, así como el diseño específico que dentro de la arquitectura nSOM se propone para su implementación.

La validación del sistema requiere que en primer lugar se aborde la descripción de un escenario real sobre el que se haya desplegado la red empleando los diseños tratados en las secciones anteriores. En este escenario se han realizado una serie de pruebas y mediciones que se detallan posteriormente para su análisis, siendo de relevancia aquellas que conciernen a la pasarela y a sus funciones de registro e interfaz de servicios, además de las relativas a la composición de servicios realizada desde un orquestador integrado en un nodo de la red.

4.2. Justificación del escenario planteado

La validación de la infraestructura planteada, y que ha sido implementada dentro del marco del proyecto Web of Objects, se lleva a cabo en un escenario de ámbito doméstico. Este escenario se empleará para demostrar el correcto funcionamiento del sistema así como para visualizar la disposición de los dispositivos que lo integran.

Se realizarán mediciones de los parámetros relacionados con la pasarela y la orquestación de servicios de acuerdo a los diseños propuestos en el capítulo anterior. Por ejemplo, se medirán los tiempos necesarios para que un servicio prestado por un dispositivo en la red de sensores pase a estar disponible una vez se reciba el mensaje HELLO; o el tiempo que se tarda en procesar una petición de servicio, desde que ésta es iniciada en la interfaz REST hasta que le es presentada la respuesta al usuario que realizó la petición.

El escenario planteado estará formado por una serie de sensores de temperatura, un sensor simulado de presencia, un nodo ofreciendo un servicio de temperatura orquestado, y un ordenador sobre se ejecutará una plataforma ESB sobre el que estará funcionando la pasarela de acceso a la WSAN, realizando tanto la función de traducción y exposición de servicios mediante una interfaz REST como la de registro de dispositivos y servicios.

4.3. Casos de uso

El estudio de los casos de uso previo a la implementación de la arquitectura nSOM en un sistema real queda descrito en el diagrama UML de la

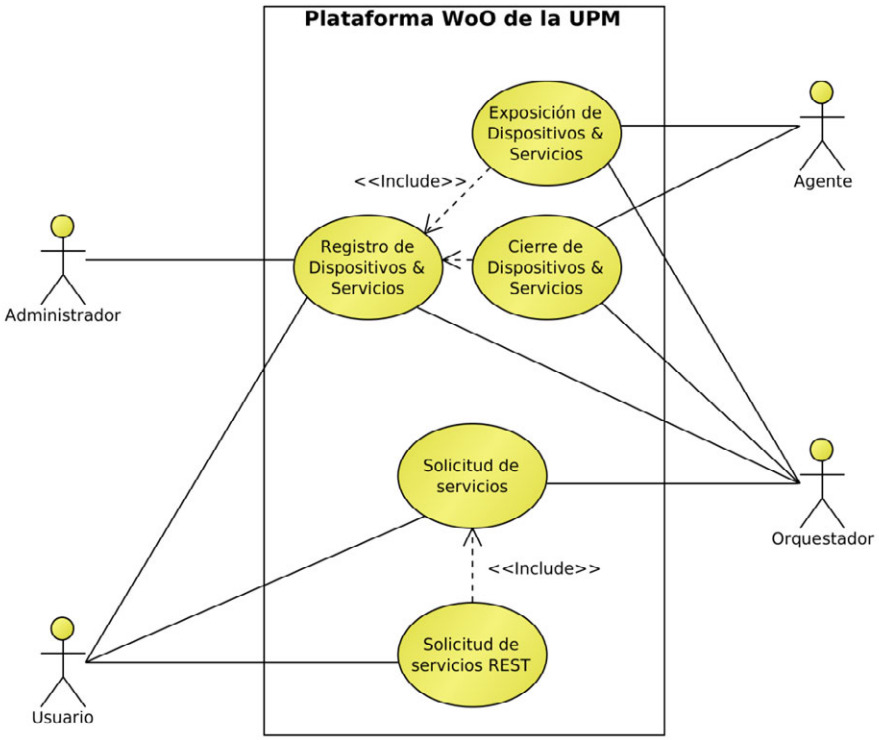

Figura 55. Casos de uso

4.3.1. Actores del sistema

Los actores que participan en el sistema son:

- **Agente.** El agente es la entidad que implementan los servicios, simple o compuesto, en los dispositivos que forman parte del sistema, y que haciendo uso del middleware nSOM los hacen disponibles mediante su exposición.
- **Orquestador.** Es la parte del sistema que realiza las peticiones al resto de servicios necesarios para componer un nuevo servicio.
- **Usuario.** Representa a cualquier entidad capaz de realizar peticiones a los servicios activos en el sistema. Puede ser un ser humano o cualquier otro tipo de entidad.
- **Administrador.** Se trata de un usuario con privilegios especiales que le habilitan para administrar el registro, y en particular poder realizar altas y bajas manuales.

4.3.2. Explicación de los casos de uso

La Tabla 34 recoge la descripción de los diferentes casos de uso expuestos en la Figura 55.

Caso de uso	Descripción	Actores
Exposición de Dispositivos & Servicios	Los dispositivos, mediante los agentes que se despliegan en ellos, se exponen a sí mismos a sus servicios. Igualmente	Agente
Registro de Dispositivos & Servicios	De un lado la exposición de dispositivos y servicios incluye su inclusión en el registro del mismo nombre. Este registro puede ser actualizado por un Administrador. Es además consultado por los usuarios del sistema, incluyendo al orquestador cuando necesita información sobre los servicios de que depende para componer un nuevo servicio.	Administrador, Usuario, Orquestador
Cierre de Dispositivos & Servicios	Los dispositivos pueden realizar un cierre controlado de sus servicios, alertando a la red de su inmediata no disponibilidad.	Agente
Solicitud de Servicios	Los usuarios, y también los orquestadores, pueden pedir servicios directamente a los agentes.	Usuario, Orquestador
Solicitud de Servicios REST	Los usuarios pueden pedir los servicios a través de una interfaz REST.	Usuario

Tabla 34. Casos de uso del sistema

4.4. Descripción del escenario

El escenario empleado para la validación del sistema expuesto en el presente proyecto simula un entorno residencial donde se emplea una red de sensores y actuadores que forman parte de un sistema más complejo siguiendo los objetivos del proyecto Web of Objects. En este escenario se dispone de un conjunto de sensores de temperatura y de presencia que ofrecen servicios web mediante una interfaz REST empleando una pasarela que ofrece la virtualización de los servicios

expuestos en la WSAN. A los sensores mencionados se les une un orquestador que ofrece un servicio compuesto basado en los resultados de las mediciones ofrecidas por los sensores de temperatura, como se muestra en la Figura 56.

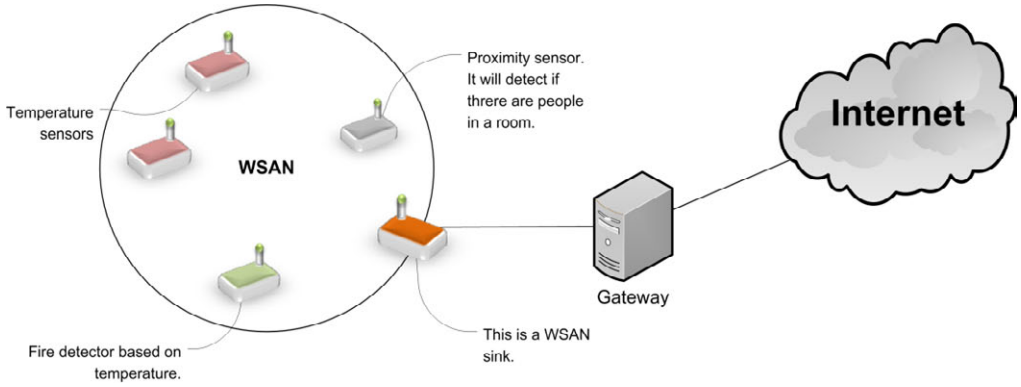

Figura 56. Escenario para la validación

Tomando como referencia la citada figura, en las siguientes subsecciones se van a describir los elementos presentes en el escenario desde las perspectivas hardware y software.

4.4.1. Elementos hardware del sistema

De un lado tenemos la red inalámbrica de sensores actuadores, compuesta por varios nodos que prestan los servicios de temperatura, presencia y un servicio compuesto basado en la temperatura. Todos los nodos presentes en la red son dispositivos SunSPOT [78], elegidos por su capacidad de procesado y almacenamiento, así como por su bajo consumo. Estos dispositivos tienen la capacidad para comunicarse inalámbricamente empleando la serie de protocolos IEEE 802.15.4, que es la tecnología de elección para las comunicaciones inalámbricas a nivel físico y de enlace en esta WSAN.

Además de los nodos, enlazando la WSAN con la pasarela se encuentra un nodo de características especiales realizando la función de sumidero. Este nodo es también un dispositivo SunSPOT, que a diferencia de los empleados como sensores carece de placa de sensores y de batería, funcionando como estación base conectada directamente al ordenador para recibir la alimentación necesaria, y servirle a éste último de medio de acceso a la WSAN.

En la Figura 57 se muestra una selección de los dispositivos empleados en el escenario. A la izquierda se encuentra el nodo sumidero, siendo los otros dos dispositivos destinados a realizar las

tareas de sensores, siendo el del centro un sensor de temperatura, y el de la derecha un sensor de presencia simulada.

Figura 57. Dispositivos SunSPOT [78] empleados.

Para la pasarela se ha empleado un miniordenador modelo ASUS EB1033 [79] dotado con un procesador Intel® Atom D2550 de doble núcleo y 4 GB de RAM, además de disponer de conectividad Wi-Fi y Ethernet entre otras especificaciones. Este ordenador cuenta con un arranque dual que lleva instalados Windows 7 Professional en su versión de 64 bits, y Ubuntu 12.04 LTS, también para 64 bits. Es precisamente sobre éste último sistema operativo sobre el que se han realizado las pruebas de validación del escenario.

Figura 58. Ordenador empleado para las funciones de pasarela

Como se ha indicado en la mención a los nodos de la WSAN, este ordenador se encuentra conectado a un sumidero SunSPOT a través de un puerto USB, adquiriendo así la capacidad de comunicarse directamente con la red de sensores.

4.4.2. Elementos software del sistema

El sistema puede describirse dividido en tres subsistemas fácilmente identificables tal y como se ilustra en la Figura 59.

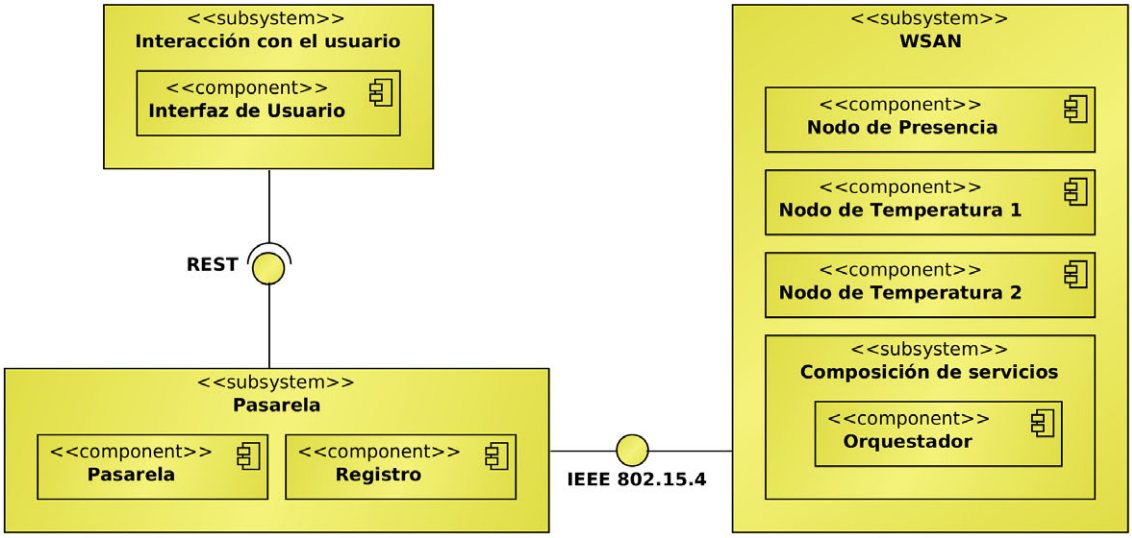

Figura 59. Susbsistemas de la arquitectura

4.4.2.1. Subsistema de interacción con el usuario

El subsistema de interacción con el usuario integra la interfaz de usuario, que esa fuera del alcance de este proyecto, pero que a efectos de validación del escenario se ha empleado un navegador web mediante el que se ha realizado el acceso a los recursos expuestos de la red utilizando la interfaz REST implementada.

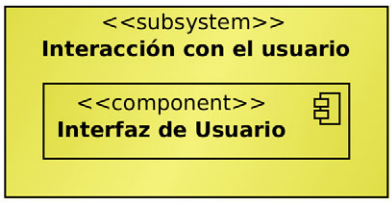

Figura 60. Subsistema «Interacción con el usuario»

4.4.2.2. Subsistema pasarela

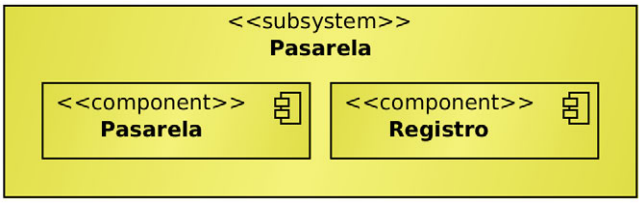

Figura 61. Subsistema «Pasarela»

El subsistema «Pasarela» está integrado por dos componentes: la pasarela propiamente dicha, y el registro de dispositivos y servicios.

La pasarela opera sobre un sistema ESB que cumple con las definiciones dadas en el Capítulo 2, sección 2.8. En particular se ha empleado la solución Fuse ESB Enterprise en su versión 7.1, desarrollada y mantenida por Red Hat, Inc., funcionando sobre Ubuntu 12.04 LTS.

El uso de una solución ESB permite homogeneizar las peticiones de los usuarios finales, empleando reglas internas que puedan adaptar las diferentes interfaces expuestas de acceso a un servicio a la requerida por el prestador el mismo.

En el caso del escenario de validación la interfaz de acceso facilitada a los usuarios es una interfaz REST accesible desde cualquier navegador a través de una URL de servicio en la forma:

http://IP-address:8181/cxf/{deviceId}/{service}

Por ejemplo, el acceso desde una red interna con dirección IP 192.168.0.20 al servicio «temperature» del dispositivo con ID 0014.4F01.0000.556B se accede desde la dirección http://192.168.0.20:8181/cxf/0014.4F01.0000.556B/temperature.

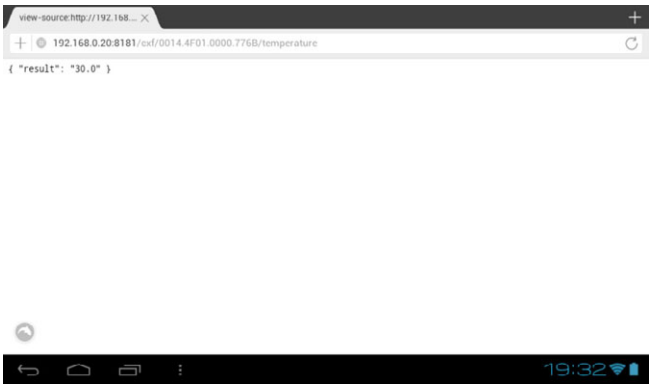

Figura 62. Acceso al servicio de temperatura desde un dispositivo Android

En cuanto al registro, se ha empleado un sistema de gestión de bases de datos MySQL en su versión 5.5.29. El acceso a la base de datos se realiza desde las aplicaciones mediante el conector JDBC, y se facilita una herramienta básica de administración del registro de dispositivos y servicios accesible mediante una interfaz REST.

La base de datos que corresponde al registro es una base de datos SQL que cumple con el diseño relacional que se describió en el Capítulo 3, sección 3.3.3.

4.4.2.3. Subsistema WSAN

El subsistema WSAN está compuesto por los nodos que forman la red de sensores. En el escenario de validación se ha contado con dos nodos de temperatura y un nodo de presencia, cuya actividad ha sido simulada mediante la acción directa sobre los componentes sensores del dispositivo.

Además este subsistema integra un subsistema de orquestación que permite la composición de nuevos servicios empleando los ya existentes en la red. En el escenario se ha contado con un nodo realizando la función de orquestación.

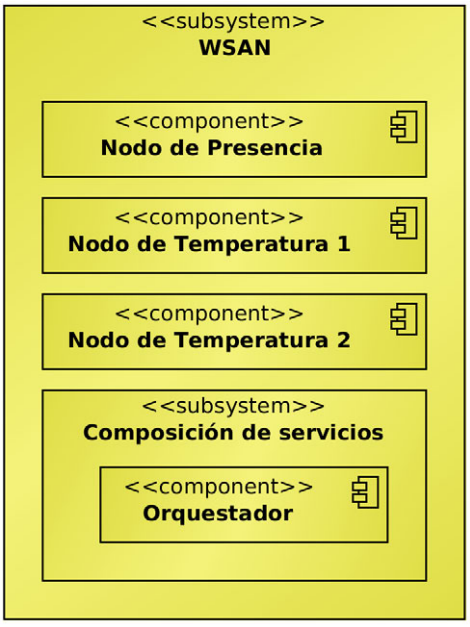

Figura 63. Subsistema «WSAN»

4.5. Análisis de resultados

Los resultados que se muestran en las siguientes secciones corresponden a diferentes pruebas realizadas sobre el escenario utilizado en la Reunión General del proyecto WoO que tuvo lugar en Barcelona durante los días 25 y 26 de junio de 2013, y que carecía de orquestador.

4.5.1. Tiempo para hacer disponible un servicio en la pasarela

Cuando un dispositivo se activa en la WSAN envía un mensaje de HELLO a todos los servicios de registro disponibles para informarles del tipo de dispositivo del que se trata y de los servicios que ofrece. En el caso del escenario que se ha propuesto, este registro se encuentra ubicado en la propia pasarela, quien además de las funciones propias de traducción de red realiza las ya citadas de registro, y las de proveedor de servicios, desplegando los componentes necesarios para intermediar entre los servicios ofrecidos por los dispositivos de la WSAN y los usuarios que emplean una interfaz REST para acceder a ellos.

4.5.1.1. Tiempo de disponibilidad

Desde un punto de vista general lo más importante es considerar el tiempo que transcurre desde la recepción del mensaje HELLO hasta que la pasarela ha desplegado el componente de servicio REST que hace las funciones de intermediario. Las siguientes pruebas se han realizado empleando los nodos sensores de temperatura y presencia, generando hasta un total de 40 envíos de HELLO que requerían del despliegue completo en el lado de la pasarela. Los tiempos obtenidos se muestran en la Tabla 35.

RESULTADOS DE LA PRUEBA «HELLO»							
Nº	Tiempo (ms)	Nº	Tiempo (ms)	Nº	Tiempo (ms)	Nº	Tiempo (ms)
1	322	**11**	325	**21**	292	**31**	324
2	289	**12**	308	**22**	267	**32**	304
3	312	**13**	290	**23**	262	**33**	248
4	343	**14**	377	**24**	312	**34**	260
5	321	**15**	260	**25**	319	**35**	236
6	301	**16**	284	**26**	292	**36**	245
7	294	**17**	323	**27**	309	**37**	307
8	309	**18**	309	**28**	259	**38**	281
9	311	**19**	295	**29**	260	**39**	289
10	295	**20**	310	**30**	283	**40**	258
Media		294,695		**Moda**		260	
Mediana		295		**Desviación Típica**		28,9054	
Mínimo		236		**Máximo**		377	

Tabla 35. Tiempo para disponibilidad de servicio tras recepción de HELLO

Destacables son los resultados de la media, situándose en los 294,695 milisegundos, que lleva asociada una desviación típica de 28,9054, siendo el valor mínimo de 236 y el máximo de 377.

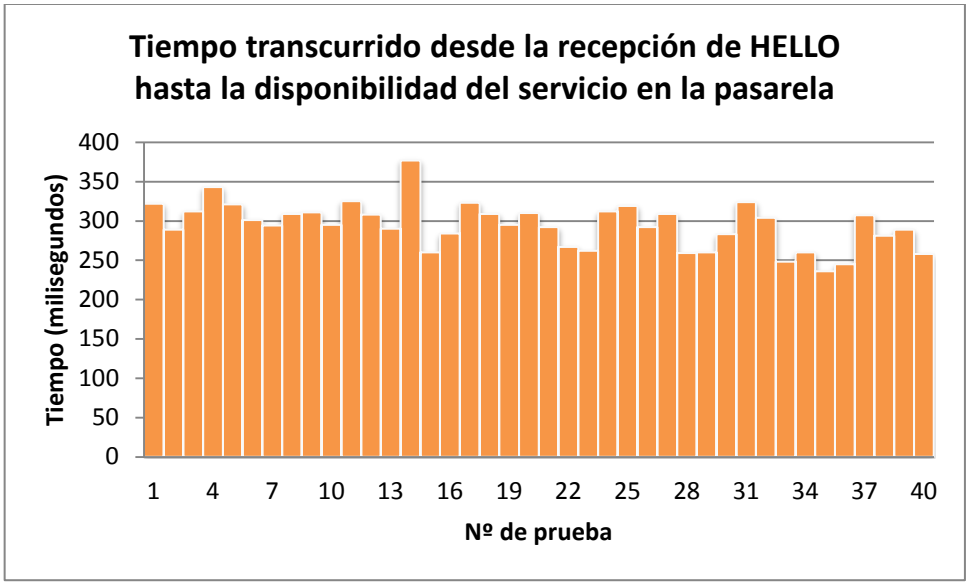

Figura 64. Gráfica de resultados de disponibilidad de servicio tras HELLO

4.5.1.2. Tiempo para el registro en la base de datos

El tiempo global requerido para que un servicio notificado desde la WSAN esté disponible en la plataforma ESB sobre todo del tiempo que se tarda en formalizar el registro. Tal y como se ha descrito con anterioridad, para este escenario se ha optado por el uso de un conocido gestor de bases de datos libre, MySQL, en su versión 5.5.29. La base de datos sigue una estructura relacional, sin hacer uso de sistemas semánticos ni de transacciones complejas. Los resultados de tiempo correspondientes al registro para las mismas peticiones analizadas en el caso anterior se recogen en la Tabla 36.

RESULTADOS DE LAS MEDICIONES DE REGISTRO EN LA BASE DE DATOS							
Nº	Tiempo (ms)	Nº	Tiempo (ms)	Nº	Tiempo (ms)	Nº	Tiempo (ms)
1	275	**11**	281	**21**	260	**31**	292
2	247	**12**	277	**22**	226	**32**	277
3	272	**13**	260	**23**	228	**33**	221
4	291	**14**	347	**24**	283	**34**	232
5	288	**15**	226	**25**	284	**35**	208
6	264	**16**	245	**26**	264	**36**	212
7	258	**17**	288	**27**	279	**37**	281
8	271	**18**	279	**28**	230	**38**	255
9	274	**19**	266	**29**	226	**39**	248
10	256	**20**	275	**30**	249	**40**	230
Media		260,625		**Moda**		226	
Mediana		264		**Desviación Típica**		27,5873	
Mínimo		208		**Máximo**		347	

Tabla 36. Tiempos de registro de entradas en la base de datos

Llaman la atención los valores tan parecidos a los necesitados para completar el proceso de disponibilidad del servicio en la pasarela, con una media de 260,625 milisegundos y una desviación típica de 27,5873.

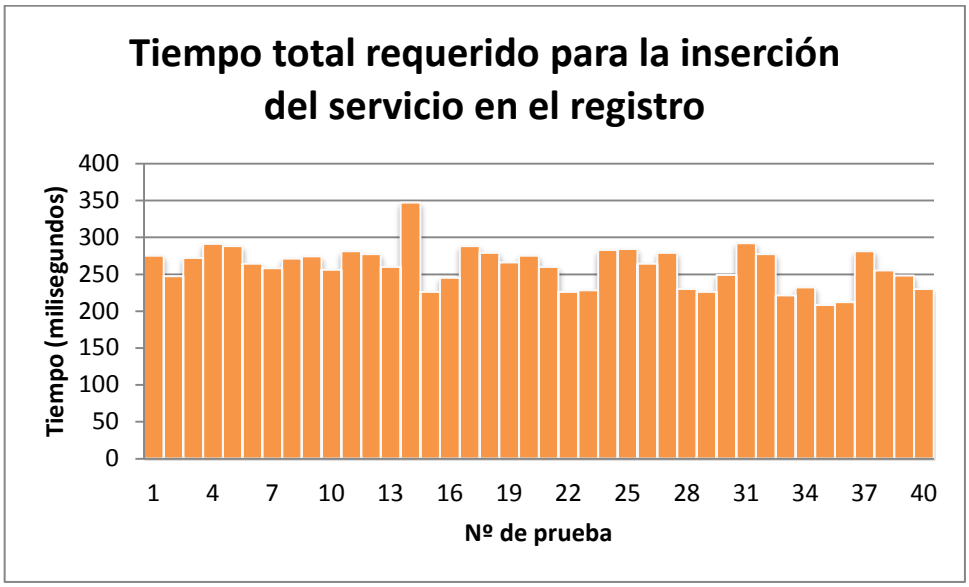

Figura 65. Gráfica de resultados para la inserción de un servicio en el registro

Si se comparan ambos resultados se puede observar que la mayor parte del tiempo destinada a desplegar un nuevo componente de servicio en el ESB corresponde precisamente al proceso de registro, resultando en un 88,41% de media, como se muestra en la Tabla 37.

COMPARATIVA DE TIEMPO TOTAL FRENTE A REGISTRO (PORCENTAJE)							
Nº	Relación (%)	Nº	Relación (%)	Nº	Relación (%)	Nº	Relación (%)
1	85,4	**11**	86,46	**21**	89,04	**31**	90,12
2	85,47	**12**	89,94	**22**	84,64	**32**	91,12
3	87,18	**13**	89,66	**23**	87,02	**33**	89,11
4	84,84	**14**	92,04	**24**	90,71	**34**	89,23
5	89,72	**15**	86,92	**25**	89,03	**35**	88,14
6	87,71	**16**	86,27	**26**	90,41	**36**	86,53
7	87,76	**17**	89,16	**27**	90,29	**37**	91,53
8	87,7	**18**	90,29	**28**	88,8	**38**	90,75
9	88,1	**19**	90,17	**29**	86,92	**39**	85,81
10	86,78	**20**	88,71	**30**	87,99	**40**	89,15
Media		88,41		**Moda**		86,92	
Mediana		88,75		**Desviación Típica**		1,91	
Mínimo		84,64		**Máximo**		92,04	

Tabla 37. Comparativa de tiempos entre disponibilidad y registro (porcentaje)

4.5.1.3. Tiempo para el despliegue del componente de servicios en el ESB

Dentro del proceso de registro existe una actividad necesaria que no está directamente implicada con la administración de la base de datos, pero sí mantiene una estrecha relación. Se trata del despliegue del componente o *bundle* en el ESB que servirá de pasarela de servicios para el nuevo

servicio dado de alta. Esta acción se lleva a cabo antes de proceder al registro, para que en el caso de que existan complicaciones se puedan tomar las medidas necesarias.

Los tiempos requeridos para el despliegue del *bundle* de servicio se recogen en la Tabla 38.

TIEMPO DE DESPLIEGUE DEL *BUNDLE* DE SERVICIO							
Nº	Tiempo (ms)	Nº	Tiempo (ms)	Nº	Tiempo (ms)	Nº	Tiempo (ms)
1	51	**11**	55	**21**	46	**31**	48
2	50	**12**	49	**22**	51	**32**	46
3	48	**13**	48	**23**	52	**33**	44
4	51	**14**	59	**24**	51	**34**	44
5	66	**15**	46	**25**	50	**35**	46
6	53	**16**	47	**26**	44	**36**	45
7	45	**17**	63	**27**	55	**37**	47
8	63	**18**	52	**28**	47	**38**	43
9	59	**19**	51	**29**	45	**39**	65
10	44	**20**	52	**30**	47	**40**	48
Media		50,4		**Moda**		51	
Mediana		48,5		**Desviación Típica**		6,0671	
Mínimo		43		**Máximo**		66	

Tabla 38. Tiempo de despliegue del *bundle* de servicio en el ESB

Son muy buenos tiempos, con una media de 50,4 milisegundos, y una desviación típica de 6,0671.

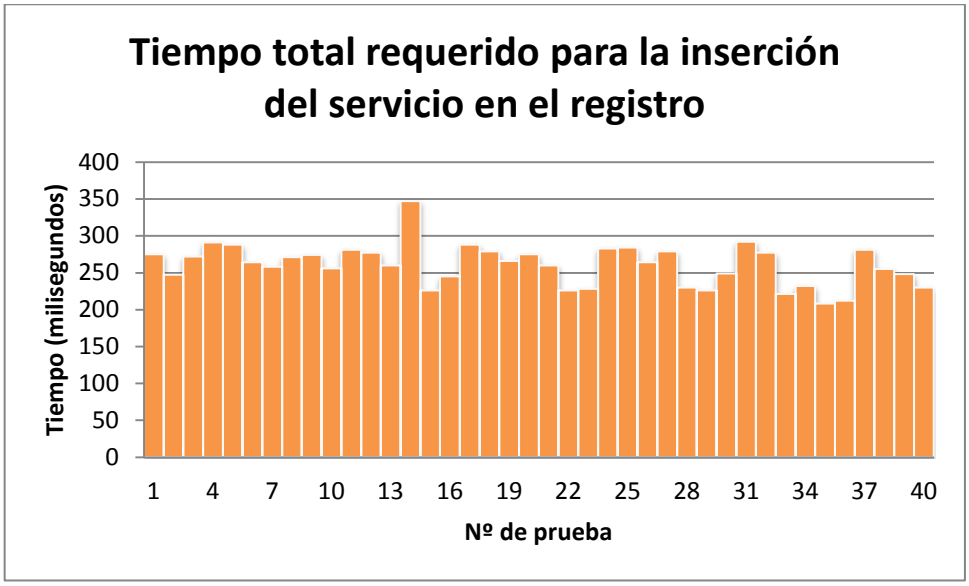

Figura 66. Tiempo requerido para insertar un servicio en el registro

A la vista de estos resultados resulta evidente que la mayor parte de la carga de proceso durante el registro le corresponde precisamente a la interacción con la base de datos. Esta es una información a tener en cuenta para futuros desarrollos, donde se podrá explorar el uso de diferentes mecanismos de registro que mejoren los tiempos de respuesta.

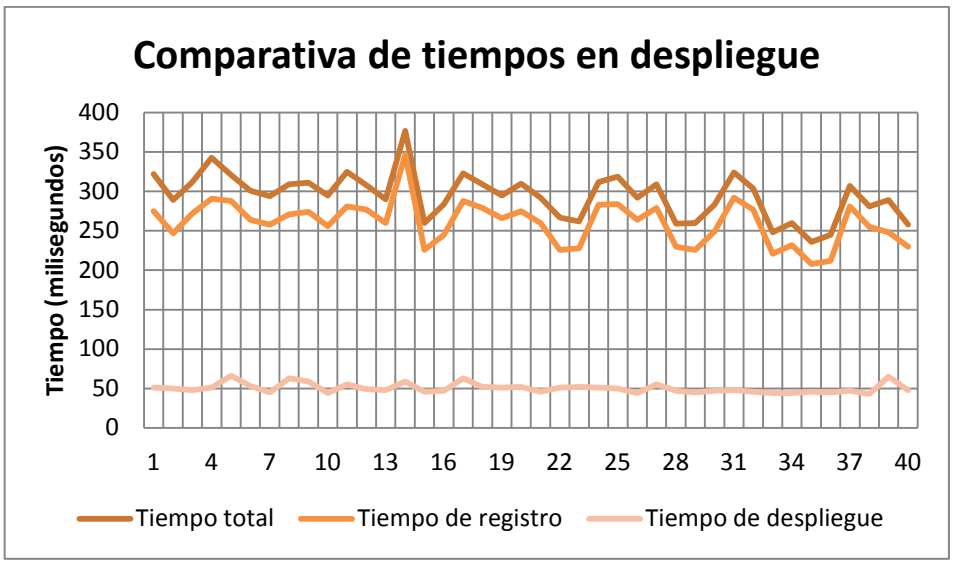

Figura 67. Comparativa de tiempos de despliegue

4.5.2. Peticiones de servicio

Cuando llega una petición de servicio realizada a través de la plataforma REST, existen dos puntos críticos de tiempo: el procesado de la petición para convertirla en el adecuado mensaje nSOM a enviar a la WSAN, y la parte correspondiente a las comunicaciones con los sensores.

4.5.2.1. Tiempo total para una petición de servicio REST

El tiempo total para una petición de servicio REST es el que transcurre desde que el *bundle* de servicio desplegado en el ESB recibe la petición de un usuario hasta que le devuelve una respuesta. Como en los casos anteriores, se han realizado 40 envíos de petición al servicio de temperatura prestado por uno de los nodos de la WSAN, obteniéndose los resultados que se muestran en la Tabla 39.

TIEMPO TOTAL DE PETICIÓN Y RESPUESTA A UN SERVICIO REST							
Nº	Tiempo (ms)	Nº	Tiempo (ms)	Nº	Tiempo (ms)	Nº	Tiempo (ms)
1	877	**11**	873	**21**	815	**31**	805
2	875	**12**	833	**22**	815	**32**	820
3	969	**13**	878	**23**	829	**33**	814
4	876	**14**	831	**24**	822	**34**	819
5	853	**15**	830	**25**	808	**35**	809
6	883	**16**	826	**26**	819	**36**	818
7	885	**17**	816	**27**	818	**37**	820
8	856	**18**	827	**28**	822	**38**	808
9	829	**19**	819	**29**	843	**39**	822
10	833	**20**	823	**30**	851	**40**	814
Media		837,075		**Moda**		819	
Mediana		824,5		**Desviación Típica**		31,9457	
Mínimo		805		**Máximo**		969	

Tabla 39. Tiempo de petición y respuesta a un servicio REST

Con un promedio de 837,075 milisegundos y una desviación típica de 31,8457 que da una dispersión del 3,82% se puede asegurar que los resultados son positivos, garantizando que salvo incidencias la respuesta llegará en un tiempo inferior a un segundo.

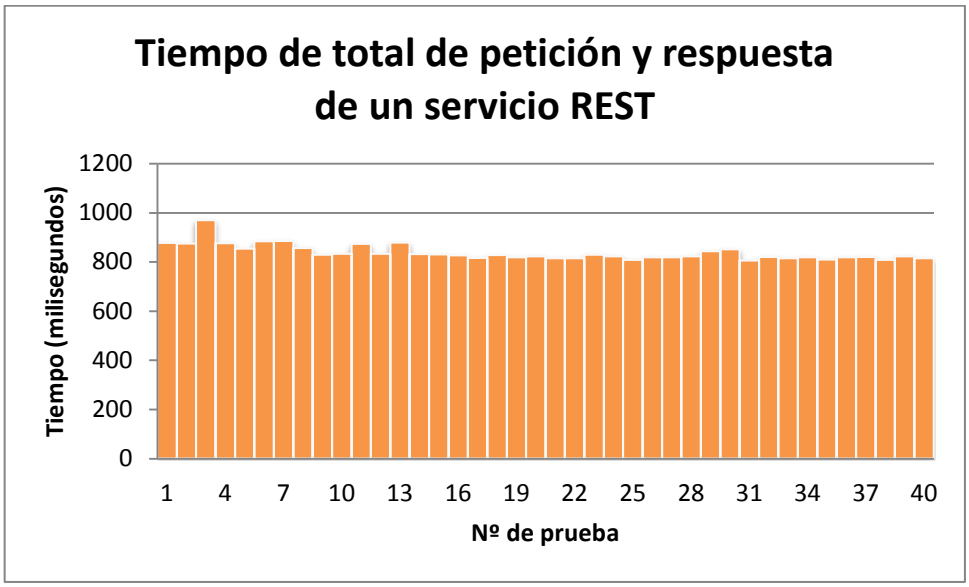

Figura 68. Tiempo total de petición y respuesta en REST

4.5.2.2. Tiempo de transmisión/recepción de un mensaje de servicio nSOM

El tiempo de transmisión y recepción de un mensaje de servicio nSOM es el que transcurre desde que se envía el radiograma desde la aplicación SunSPOTHost en el DIAL de la pasarela, hasta que se recibe el correspondiente radiograma de respuesta. Los resultados se han obtenido empleando las mismas pruebas que en la sección anterior, para mantener la correlación, obteniéndose los valores que se recogen en la Tabla 40.

TIEMPO TOTAL DE TRANSMISIÓN/RECEPCIÓN DE UNA SOLICITUD DE SERVICIO							
Nº	Tiempo (ms)	Nº	Tiempo (ms)	Nº	Tiempo (ms)	Nº	Tiempo (ms)
1	728	**11**	736	**21**	725	**31**	724
2	732	**12**	726	**22**	726	**32**	726
3	726	**13**	727	**23**	734	**33**	726
4	727	**14**	732	**24**	731	**34**	726
5	728	**15**	731	**25**	725	**35**	725
6	733	**16**	726	**26**	727	**36**	725
7	728	**17**	728	**27**	728	**37**	729
8	729	**18**	728	**28**	730	**38**	724
9	729	**19**	732	**29**	731	**39**	725
10	732	**20**	727	**30**	726	**40**	726
Media		728,1		**Moda**		726	
Mediana		727,5		**Desviación Típica**		2,95	
Mínimo		724		**Máximo**		736	

Tabla 40. Tiempo de transmisión/recepción para una solicitud de servicio

Los datos obtenidos arrojan una media de 728,1 milisegundos, con una desviación típica de 2,95, lo que como en la medición anterior arroja unos resultados bastante estables. Un dato interesante que puede extraerse de la comparación de ambas pruebas es que el tiempo de transmisión y recepción representa un promedio del 87% con respecto del tiempo total necesitado para atender la petición invocada desde la interfaz REST.

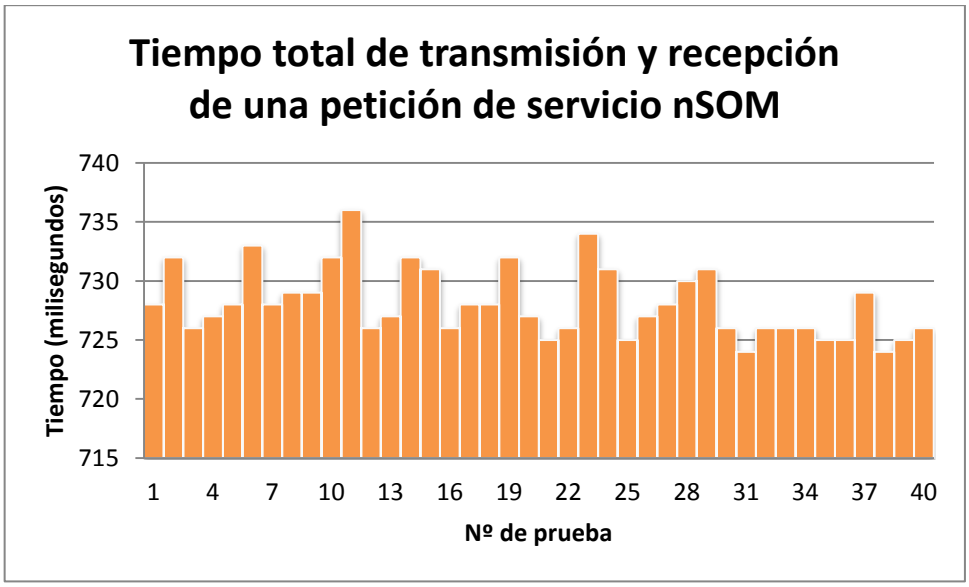

Figura 69. Tiempo de transmisión y recepción de una petición nSOM

4.5.3. Cierre ordenado de un dispositivo y sus servicios.

El cierre ordenado de un dispositivo y de sus servicios asociados se produce cuando antes de apagarse se envía un mensaje tipo BYE al registro. De esta forma el registro puede proceder a eliminar la entrada o marcarla como no disponible, siendo usada la primera opción en el escenario usado en la validación.

Como en los experimentos anteriores, se han realizado 40 peticiones de cierre de dispositivo y servicio, obteniéndose los resultados que se describen en las siguientes subsecciones.

4.5.3.1. Tiempo total de cierre del servicio en la pasarela

El tiempo total de cierre de servicio en la pasarela es el tiempo que transcurre desde que se recibe el mensaje tipo BYE hasta que el servicio ha sido eficazmente eliminado del registro, y el correspondiente *bundle* de servicio ha sido desconectado del ESB. Los resultados se muestran en la Tabla 41.

TIEMPO TOTAL DE CIERRE DEL SERVICIO EN LA PASARELA							
Nº	Tiempo (ms)	Nº	Tiempo (ms)	Nº	Tiempo (ms)	Nº	Tiempo (ms)
1	226	**11**	255	**21**	244	**31**	282
2	241	**12**	546	**22**	231	**32**	218
3	292	**13**	280	**23**	258	**33**	223
4	295	**14**	232	**24**	294	**34**	204
5	228	**15**	245	**25**	217	**35**	194
6	244	**16**	301	**26**	247	**36**	220
7	323	**17**	310	**27**	219	**37**	244
8	237	**18**	259	**28**	225	**38**	231
9	267	**19**	300	**29**	206	**39**	217
10	311	**20**	266	**30**	276	**40**	212
Media		258		**Moda**		244	
Mediana		244		**Desviación Típica**		57,60	
Mínimo		194		**Máximo**		546	

Tabla 41. Tiempo total de cierre del servicio en la pasarela

Los tiempos que se han obtenido indican una media de 258 milisegundos con una desviación típica de 57,60, existiendo valores muy dispares, como reflejan la existencia de un valor mínimo de 194 milisegundos, y un máximo de 546.

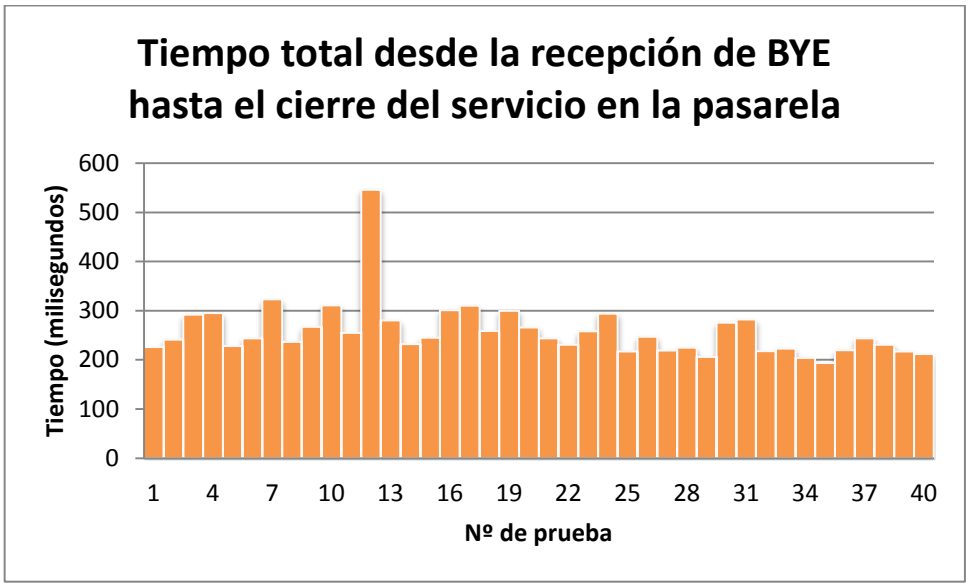

Figura 70. Tiempo total de cierre de un servicio en la pasarela

4.5.3.2. Tiempo de borrado del servicio en el registro

En la misma prueba que la anterior se ha medido adicionalmente el tiempo que tarda en borrarse la entrada del registro correspondiente a un servicio que ha anunciado su despedida. Una vez más parece razonable concluir que la parte que mayor tiempo añade a la acción sigue siendo la que corresponde a la interacción con el sistema de gestión de la base de datos. Los datos se recogen en la Tabla 42.

TIEMPO TOTAL DE BORRADO DE SERVICIOS EN EL REGISTRO							
Nº	Tiempo (ms)	Nº	Tiempo (ms)	Nº	Tiempo (ms)	Nº	Tiempo (ms)
1	189	**11**	232	**21**	220	**31**	254
2	213	**12**	518	**22**	205	**32**	196
3	258	**13**	238	**23**	235	**33**	202
4	267	**14**	205	**24**	271	**34**	184
5	201	**15**	218	**25**	194	**35**	171
6	215	**16**	278	**26**	226	**36**	198
7	298	**17**	284	**27**	196	**37**	223
8	211	**18**	236	**28**	204	**38**	210
9	224	**19**	276	**29**	183	**39**	192
10	278	**20**	227	**30**	253	**40**	190
Media		231,825		**Moda**		278	
Mediana		219		**Desviación Típica**		56,50	
Mínimo		171		**Máximo**		518	

Tabla 42. Tiempo total de borrado de un servicio del registro

Con un promedio de 231,825 milisegundos y una desviación típica de 56,50 se aprecia una evidente relación entre los tiempos requeridos para la eliminación de una entrada en el registro, y el tiempo total requerido para dar de baja un servicio en el sistema. Ante estos resultados parece evidente que el mayor retardo en el sistema lo introduce el acceso a la base de datos empleada para el registro.

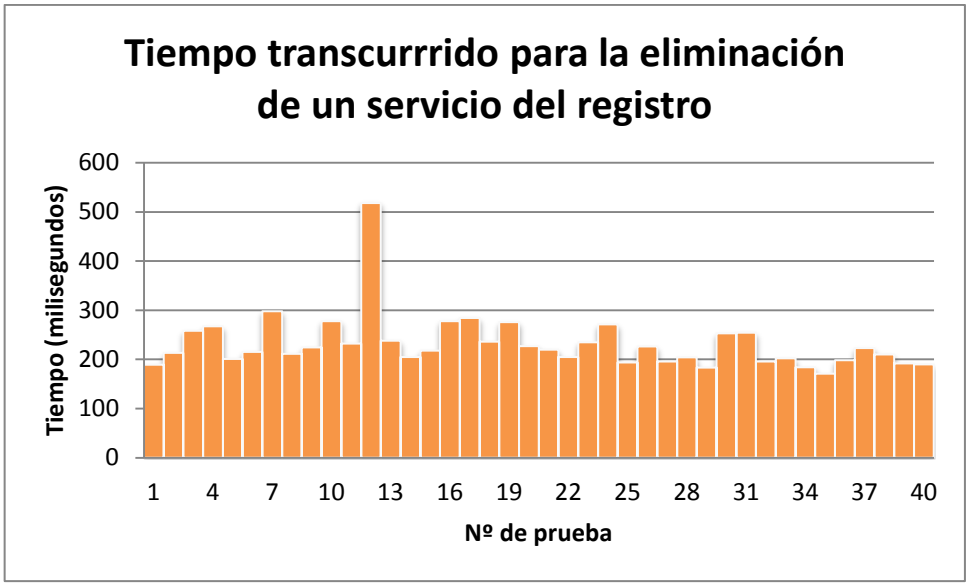

Figura 71. Tiempo total para borrar un servicio del registro

4.5.4. Uso de memoria de los componentes de servicio en el ESB

Se ha realizado a su vez una medición de la carga de memoria requerida por el ESB que opera en la pasarela para asumir los nuevos componentes de servicio que se despliegan en él. El proceso para esta medición se realiza con un *script* de sistema que genera un nuevo *bundle* a partir de los arquetipos diseñados para que pueda ser desplegado sobre el ESB. El mismo *script* lo despliega en

caliente sobre el bus y obtiene un mapa de la asignación de memoria del proceso que corresponde al bus, comparándolo con los anteriores y devolviendo la diferencia.

El mapa de asignación de memoria se obtiene ejecutando la instrucción *pmap* del sistema operativo. Otras opciones, como *ps* dan información sobre la memoria que el sistema operativo reserva para el proceso, con lo que el dato facilitado no se correspondería con la realidad de la carga real de memoria que se está empleando.

En la gráfica que se muestra en la Figura 72 se recogen los datos de la prueba realizada. El consumo de memoria permanece estable durante toda la prueba, salvo en el momento del despliegue del *bundle* de prueba nº 37, en el que se produce un salto no directamente relacionable a tal acción. Durante el resto de la prueba, no apreciable en la gráfica, la carga de memoria en uso por el ESB permanece estable, salvo en el citado salto, manteniendo la misma cantidad de memoria. Es en la memoria asignada donde se van produciendo pequeños incrementos de 4 kilobytes por cada *bundle* de servicio desplegado.

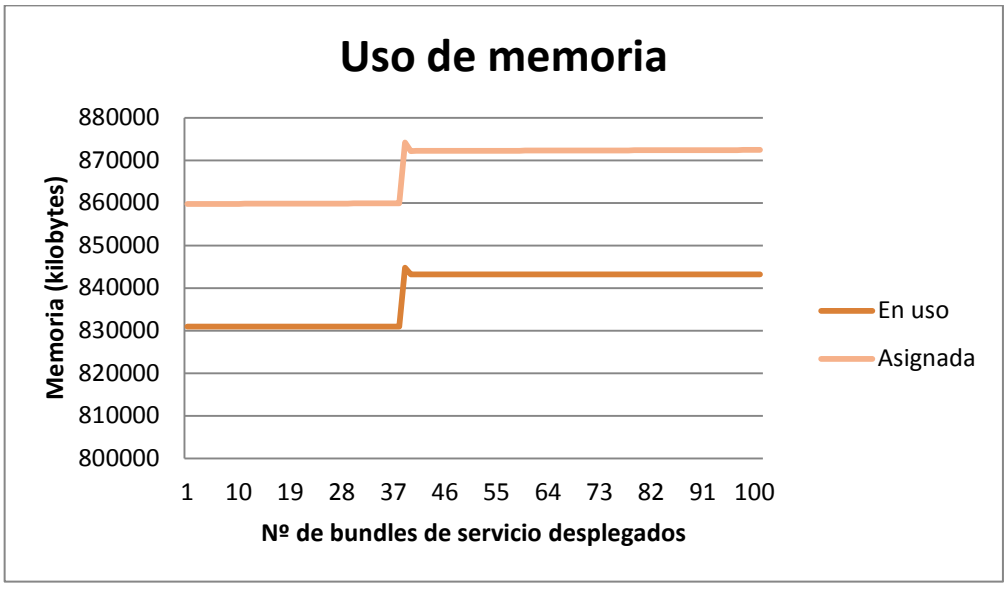

Figura 72. Gráfica sobre el uso de memoria en el ESB

Capítulo 5

Conclusiones y trabajos futuros

5.1. Conclusiones

Esta memoria se ha centrado en tres bloques principales:

a) En el primer bloque se ha realizado un estudio del estado del arte en relación a las tecnologías que resultan de interés para afrontar los problemas que pueden encontrarse en el desarrollo del proyecto.
b) En el segundo bloque se describe la especificación técnica de la arquitectura nSOM desarrollada en el GRyS junto a las actualizaciones realizadas en el marco de este proyecto.
c) Finalmente en el tercer y último bloque se pone en práctica la solución técnica en un escenario real para su validación mediante el análisis de diferentes métricas de servicio.

Para finalizar esta memoria, y antes de abordar posibles desarrollos futuros basados en la experiencia acumulada durante la realización de este trabajo, se va a proceder a un breve resumen de los temas que se han tratado en los diferentes capítulos.

En el **Capítulo 1** se ha realizado una introducción a los conceptos sobre los que se encuadra este proyecto, abordando la definición de la Ciudad del Futuro y de Internet de las Cosas, de los objetivos del proyecto Web of Objects sobre el que se ha desarrollado este trabajo, y la relación que todos estos conceptos e intereses tecnológicos tienen con proyectos de la relevancia de Europa 2020. Posteriormente se han definido los objetivos concretos de este Proyecto Fin de Carrera, y la organización de los contenidos en esta memoria, para finalizar con una mención al contexto en el que está enmarcado.

En el **Capítulo 2** se ha realizado un extenso estudio del estado del arte aplicable a Internet de las Cosas y a los objetivos del proyecto. Se ha profundizado en los conceptos tecnológicos relacionados con las Redes Inalámbricas de Sensores y Actuadores (WSAN), con la comunicación Máquina a Máquina y fundamentalmente con la arquitectura de referencia para Internet de las Cosas.

A continuación se han tratado las diferentes tecnologías de red que pueden encontrarse en escenarios típicos propios de los objetivos de este proyecto, incluyendo topologías y protocolos habituales en el ámbito de los dispositivos con recursos limitados.

Seguidamente se han abordado las tecnologías propias del nivel de aplicación de uso en el ámbito de la web, al ser este el entorno donde se enmarca la exposición de los servicios en este proyecto. Se han descrito el protocolo HTTP específico de las comunicaciones web, y CoAP, adaptado a entornos con limitaciones. De igual forma se han detallado las técnicas empleadas para la representación de datos, y las arquitecturas de desarrollo propias de los servicios web.

Posteriormente se han mencionado los principios de la arquitectura de software orientada a servicios (SOA) así como otras soluciones de desarrollo comunes en aplicaciones distribuidas.

Para finalizar, se ha definido el concepto de composición de servicios, y se han descrito los diferentes mecanismos posibles, así como los lenguajes normalizados para cada mecanismo.

El **Capítulo 3** comienza con la descripción de la arquitectura nSOM desarrollada en el Grupo de Redes y Servicios de Próxima Generación. Una vez expuesta la arquitectura y descritas las funciones básicas de sus componentes, se realiza una propuesta de proyección sobre la arquitectura de referencia IoT-A, mostrando que la arquitectura nSOM es compatible con el modelo propuesto.

A continuación se describe el modelo de comunicaciones para la composición de un servicio de tipo orquestado en una red basada en nSOM. Se describen los intercambios de mensajes entre los elementos de la red en dos escenarios posibles. En un primer escenario se trata la composición de un servicio orientado a consulta basado en el consumo de servicios simples. Mientras que en el segundo escenario el servicio compuesto sigue un modelo más complejo en el que es consumidor de servicios simples y servicios orientados a eventos, siendo capaz de generar nuevos eventos en función de los resultados de las consultas realizadas a los servicios simples, así como de la llegada de nuevos eventos a los que se encuentre suscrito.

Le sigue la descripción del diseño de una pasarela para la **virtualización de dispositivos** en una red con recursos limitados como es una WSAN, empleando para ello los servicios que proporcionan. Esta pasarela incluye un registro de dispositivos y servicios inspirada en la propuesta por la arquitectura de referencia IoT-A, utilizando una base de datos relacional.

Finalmente se describe el diseño de una propuesta para un motor de orquestación capaz de componer nuevos servicios virtuales empleando otros existentes.

Por último en el **Capítulo 4** se describe un escenario concreto sobre el que se han aplicado las propuestas de **virtualización de dispositivos** descritas en el capítulo anterior, dentro del marco del proyecto europeo Web of Objects. De este escenario se realiza un estudio de los casos de uso, y

de los componentes que forman parte de él, tanto en lo que respecta al hardware utilizado en las pruebas como al software desarrollado.

A este escenario se le ha sometido a diferentes pruebas de validación, comprobando el tiempo necesario para que un servicio esté disponible, los tiempos de respuesta ante las peticiones de servicio, y la carga del sistema ante una simulación de múltiples dispositivos, en orden de conocer la **capacidad** del sistema, y su potencial **escalabilidad**.

Tras los resultados obtenidos, se puede concluir que el diseño propuesto cumple con los objetivos propuestos en la introducción de esta memoria, ofreciendo una pasarela de servicios capaz de **virtualizar los dispositivos** de una WSAN en un entorno IoT-A.

5.2. Trabajos futuros

En base a la experiencia adquirida durante el desarrollo de este proyecto, se han considerado las siguientes ideas como posibles trabajos futuros sobre el sistema descrito, considerándose útiles tanto en su optimización como en su mejora:

- En relación a la composición de servicios:
  - Explorar el desarrollo de un lenguaje reducido de orquestación de servicios que pueda ser implementado por completo en el motor de orquestación, permitiendo la creación de servicios complejos multihilo, la composición semántica apoyada en servicios de descubrimiento. Este tipo de motor puede ofrecer ventajas de cara a la **sostenibilidad de la composición**, así como ofrecer mecanismos para la **tolerancia a fallos** en la red.
  - Explorar el modelo de **composición por coreografía** para su uso en redes con recursos limitados, y particularmente en WSAN. Este modelo puede emplearse, por ejemplo, para la autoconfiguración de los dispositivos.
  - Desarrollar una **aplicación de gestión de composiciones** que emplee una interfaz de usuario amigable para la composición de servicios.
- En relación a la virtualización de servicios:
  - Estudiar posibles mejoras en el uso del registro de dispositivos y servicios, teniendo por objetivo la **reducción de los tiempos** destinados al acceso al mismo, que como se ha visto en el Capítulo 4, sección 4.5, tiene una elevada influencia en las acciones que dependen de él.

  - Estudiar mejoras en el esquema de despliegue de los componentes de servicio, siguiendo una aproximación al modelo de entidades virtuales propuesto en la arquitectura de referencia IoT-A.
  - Estudiar el consumo de servicios **en sentido inverso** desde la WSAN a servicios remotos expuestos en la pasarela. Por ejemplo, el uso de un servicio de orquestación integrado en una WSAN que emplee un servicio externo a ella para añadir valor a su composición.
  - Estudiar el uso de la pasarela en la creación de **túneles entre varias WSAN**, operando entre ellas como si fueran una sola.
  - Establecer un **sistema de almacenaje temporal** (cache) que permita reducir las consultas realizadas a los dispositivos en una WSAN, con la consiguiente reducción de tráfico en la WSAN, y el más que posible ahorro energético.
- En relación a la arquitectura nSOM:
  - Analizar la **integración de las funciones de administración** descritas en la arquitectura IoT-A que no tienen correspondencia en el modelo de componentes de nSOM.
  - Explorar el uso de mensajes tipo Multicast, Anycast, y Reverse Unicast definidos en la arquitectura de referencia IoT-A, adicionales a los mensajes tipo Unicast y Broadcast ya implementados en la plataforma.
- En relación a las WSAN:
  - Explorar, diseñar e implementar **topologías de red basadas en clusters** organizador por brokers dotados de capacidad de registro y despacho. Este tipo de topologías podría facilitar el despliegue en redes ampliamente pobladas por dispositivos, reduciendo el consumo necesario para el encaminamiento de los mensajes empleando algoritmos optimizados.
  - Basada en la anterior propuesta, explorar un modelo para WSAN extensas inspirado en los sistemas celulares.

Anexo I

API de componentes ESB

Interface RmiInterface

org.upm.sunspot.host

All Superinterfaces:

Remote

All Known Implementing Classes:

SunSPOTHost

```
public interface RmiInterface
extends Remote
```

Method Summary	
String	**getLocalMACAddress**()
byte[]	**sendToWSAN**(byte[] message, String address)

Method Detail

sendToWSAN

```
byte[] sendToWSAN(byte[] message,
                  String address)
           throws RemoteException
```

Throws:

RemoteException

---

getLocalMACAddress

```
String getLocalMACAddress()
                   throws RemoteException
```

Throws:

RemoteException

Class SunSPOTHost

org.upm.sunspot.host

java.lang.Object

└ **org.upm.sunspot.host.SunSPOTHost**

All Implemented Interfaces:

Remote, RmiInterface

---

public class **SunSPOTHost**

extends Object

implements RmiInterface

Since:

2013-06-17

Constructor Summary
SunSPOTHost()

Method Summary	
String	**getLocalMACAddress**() Get the MAC address for the local IEEE 802.15.4 interface.
static void	**main**(String[] args)
byte[]	**sendToWSAN**(byte[] message, String address) Sends a request to the WSAN.

Constructor Detail

SunSPOTHost

public **SunSPOTHost**()

Method Detail

getLocalMACAddress

public String **getLocalMACAddress**()

Get the MAC address for the local IEEE 802.15.4 interface.

Specified by:

getLocalMACAddress in interface RmiInterface

Returns:

The dotted IEEE 802.15.4 MAC address.

---

sendToWSAN

```
public byte[] sendToWSAN(byte[] message,
                         String address)
```

Sends a request to the WSAN.

Specified by:

`sendToWSAN` in interface `RmiInterface`

Parameters:

`message` - The message to be sent to the WSAN.

`address` - The destination address in IEEE 802.15.4 MAC format.

Returns:

The response for the request sent to the WSAN.

---

main

```
public static void main(String[] args)
                throws Exception
```

Parameters:

`args` - Command line arguments.

Throws:

`Exception`

Interface nSOMRestService

org.upm.nsom

All Known Implementing Classes:

nSOM_onESB

```
public interface nSOMRestService
```

Method Summary	
Response	**addIncomingBroadcastMessage**(byte[] in)
Response	**getService**(String deviceId, String resource)

Method Detail

getService

```
Response getService(String deviceId, String resource)
```

---

addIncomingBroadcastMessage

```
Response addIncomingBroadcastMessage(byte[] in)
```

Class nSOM_onESB

org.upm.nsom

java.lang.Object

└ **org.upm.nsom.nSOM_onESB**

All Implemented Interfaces:

nSOMRestService

---

public class **nSOM_onOSGi**

extends Object

implements nSOMRestService

The core nSOM engine for the ESB. This class provides de core functions for the interaction with nSOM capable services, making the protocol translation between requests from the web interface, responses from the WSAN, and the HELLO and BYE messages originated by the nodes in the WSAN.

Constructor Summary
nSOM_onESB()

Method Summary	
Response	**addIncomingBroadcastMessage**(byte[] in) This method acts as a listener from incoming HELLO and BYE messages from the WSAN.
Response	**getService**(String deviceId, String resource) The method to be invoked when a request is made from the web interface.
String	**parseResponseFromWSAN**(String responseFromWSAN) This method parses the message received from the WSAN as a response for a previous petition.

Constructor Detail

nSOM_onESB

public **nSOM_onESB**()

Method Detail

getService

public Response **getService**(String deviceId, String resource)

The method to be invoked when a request is made from the web interface.

Specified by:

getService in interface nSOMRestService

Parameters:

deviceId - The identification for the device associated with the service.

resource - The resource that maps the service that is invoked.

Returns:

The parsed response for the request if Ok, ERROR otherwise.

---

parseResponseFromWSAN

`public String` **`parseResponseFromWSAN`**`(String responseFromWSAN)`

This method parses the message received from the WSAN as a response for a previous petition.

Parameters:

`responseFromWSAN` - The message form the WSAN.

Returns:

The parsed conent.

---

addIncomingBroadcastMessage

`public Response` **`addIncomingBroadcastMessage`**`(byte[] in)`

This method acts as a listener from incoming HELLO and BYE messages from the WSAN.

Specified by:

`addIncomingBroadcastMessage` in interface `nSOMRestService`

Parameters:

`in` - The incoming nSOM PDU.

Returns:

Ok if the incoming message is parsed correctly, ERROR otherwise.

Interface RestRegistry

org.upm.woo.wooreg

All Known Implementing Classes:

WoORegistry

```
public interface RestRegistry
```

Method Summary	
Response	**addRegistryEntry**(String deviceMAC, String deviceId, String deviceType, String deviceUbication, String deviceLocation, String serviceName, String serviceOperation, String serviceDescription, String serviceUrl)
Response	**deleteRegistryEntry**(String deviceId)
Response	**getRegistryEntry**(UriInfo info)
String	**getServiceName**(String invocation)
String	**getServiceOperation**(String invocation)

Method Detail

getRegistryEntry

```
Response getRegistryEntry(UriInfo info)
```

---

addRegistryEntry

```
Response addRegistryEntry(String deviceMAC,
                          String deviceId,
                          String deviceType,
                          String deviceUbication,
                          String deviceLocation,
                          String serviceName,
                          String serviceOperation,
                          String serviceDescription,
                          String serviceUrl)
```

---

deleteRegistryEntry

```
Response deleteRegistryEntry(String deviceId)
```

---

getServiceOperation

```
String getServiceOperation(String invocation)
```

---

getServiceName

```
String getServiceName(String invocation)
```

Class WoORegistry

org.upm.woo.wooreg

java.lang.Object

└ **org.upm.woo.wooreg.WoORegistry**

All Implemented Interfaces:

RestRegistry

---

public class **WoORegistry**

extends Object

implements RestRegistry

This class implements a Devices and Service Registry using a SQL database.

Constructor Summary
WoORegistry()

Method Summary	
Response	**addRegistryEntry**(String deviceMAC, String deviceId, String deviceType, String deviceUbication, String deviceLocation, String serviceName, String serviceOperation, String serviceDescription, String serviceUrl) Adds a new entry to the Registry
Response	**deleteRegistryEntry**(String deviceId) Removes an entry from the Registry.
Response	**getRegistryEntry**(UriInfo info) This method provides the way to retrieve information about an entry in the Registry from a REST interface.
String	**getServiceName**(String invocation)
String	**getServiceOperation**(String invocation)

Constructor Detail

WoORegistry

public **WoORegistry**()

Method Detail

getRegistryEntry

public Response **getRegistryEntry**(UriInfo info)

This method provides the way to retrieve information about an entry in the Registry from a REST interface.

Specified by:

getRegistryEntry in interface RestRegistry

Parameters:

info - The URL associated information to retrieve the query with the parameters regarding the entry requested.

Returns:

Ok if the access to the Registry was succesful, ERROR otherwise.

---

addRegistryEntry

```
public Response addRegistryEntry(String deviceMAC,
                                 String deviceId,
                                 String deviceType,
                                 String deviceUbication,
                                 String deviceLocation,
                                 String serviceName,
                                 String serviceOperation,
                                 String serviceDescription,
                                 String serviceUrl)
```

Adds a new entry to the Registry

Specified by:

`addRegistryEntry` in interface `RestRegistry`

Parameters:

`deviceMAC` - The MAC of the device.

`deviceId` - The ID of the device.

`deviceType` - The device type.

`deviceUbication` - The device ubication.

`deviceLocation` - The device relative location, that is, local or remote.

`serviceName` - The service name.

`serviceOperation` - The service operation.

`serviceDescription` - The service description.

`serviceUrl` - The entrypoint for the service as an URL.

Returns:

Ok if the insertion was succesful, ERROR otherwise.

---

deleteRegistryEntry

```
public Response deleteRegistryEntry(String deviceId)
```

Removes an entry from the Registry.

Specified by:

`deleteRegistryEntry` in interface `RestRegistry`

Parameters:

`deviceId` - The identification of the device to be removed.

Returns:

Ok if the operation was succesful, ERROR, otherwise.

Interface RestService

org.upm.woo.temperature

All Known Implementing Classes:

RestTemperature

public interface **RestService**

Method Summary	
Response	**getService**(UriInfo info)

Method Detail

getService

Response **getService**(UriInfo info)

Class RestTemperature

org.upm.woo.temperature

java.lang.Object

└ **org.upm.woo.temperature.RestTemperature**

All Implemented Interfaces:

RestService

---

public class **RestTemperature**

extends Object

implements RestService

This class represents the archetype for a REST interface for temperature services registered in the WSAN.

Constructor Summary
RestTemperature()

Method Summary	
Response	**getService**(UriInfo info) Proccess the HTTP GET method invocation

Constructor Detail

RestTemperature

public **RestTemperature**()

Method Detail

getService

public Response **getService**(UriInfo info)

Proccess the HTTP GET method invocation

Specified by:

getService in interface RestService

Parameters:

info - Information about the URI used in the request.

Returns:

The response to the request if Ok, ERROR otherwise

Class RestPresence

org.upm.woo.presence

java.lang.Object

└ **org.upm.woo.presence.RestPresence**

All Implemented Interfaces:

RestService

---

public class **RestPresence**

extends Object

implements RestService

This class represents the archetype for a REST interface for presence services registered in the WSAN.

Constructor Summary
RestPresence()

Method Summary	
Response	**getService**(UriInfo info) Proccess the HTTP GET method invocation

Constructor Detail

RestPresence

public **RestPresence**()

This class represents the archetype for a REST interface for presence services registered in the WSAN.

Method Detail

getService

public Response **getService**(UriInfo info)

Proccess the HTTP GET method invocation

Specified by:

getService in interface RestService

Parameters:

info - Information about the URI used in the request.

Returns:

The response to the request if Ok, ERROR otherwise

Anexo II

API de orquestación nSOM

Interface Orchestrator

org.upm.nsom.framework.Orchestrator

All Known Implementing Classes:

OrchestratorAgent

`public interface` **`Orchestrator`**

Provides the way for the implementors to be notified by an OrchestratorEngine whenever it generates or consumes an event that is expected in the listener identified by orchestrationId.

Method Summary	
`void`	**`orchestrationListener`**`(int orchestrationId, String event)` Notifies an event to the listener.

Method Detail

orchestrationListener

```
void orchestrationListener(int orchestrationId,
                           String event)
```

Notifies an event to the listener.

Parameters:

`orchestrationId` - Identity of the orchestration assciated to the event.

`event` - Event.

Class OrchestratorAgent

org.upm.nsom.agents

```
java.lang.Object
  └ org.upm.nsom.agents.OrchestratorAgent
```

All Implemented Interfaces:

IAgent, Orchestrator

```
public class OrchestratorAgent
extends Object
implements IAgent, Orchestrator
```

An agent that uses a set of rules to orchestrate a service using the OrchestrationEngine from the nSOM framework

---

Constructor Summary
OrchestratorAgent()

Method Summary	
String	**getnSOMAgentDescription**() Obtain the nSOMAgentDescription.
nSOMServiceAgentObject	**getServiceAgentObject**() Obtain the service description object for this Agent.
void	**load**(nSOMContext aNSOMContext) Performs the load process of the Agent in the Service Execution Platform.
void	**orchestrationListener**(int orchestrationId, String event) Notifies an event to the listener.
void	**receptacle**(ServiceContext aOperationInvocation) Implements the receptacle of the Agent.
void	**run**() Performs the run process of the Agent and the service registering in the Registry.
Void	**stop**() Performs the stop process of the Agent and the service unregistering in the Registry.
Void	**unload**() Performs the unload process of the nSOMAgent in the Service Execution Platform.

Constructor Detail

OrchestratorAgent

```
public OrchestratorAgent()
```

Method Detail

getnSOMAgentDescription

`public String` **`getnSOMAgentDescription`**`()`

Obtain the nSOMAgentDescription. This method provides the name of the agent which have been invoked.

Specified by:

`getnSOMAgentDescription` in interface `IAgent`

Returns:

String with the agent description

---

getServiceAgentObject

`public nSOMServiceAgentObject` **`getServiceAgentObject`**`()`

Obtain the service description object for this Agent.

Specified by:

`getServiceAgentObject` in interface `IAgent`

Returns:

nSOMServiceAgentObject of the Agent

---

load

`public void` **`load`**`(nSOMContext aNSOMContext)`

Performs the load process of the Agent in the Service Execution Platform.

Specified by:

`load` in interface `IAgent`

---

receptacle

`public void` **`receptacle`**`(ServiceContext aOperationInvocation)`

Implements the receptacle of the Agent. This method is invoked to pass the service invocations to the agents.

Specified by:

`receptacle` in interface `IAgent`

---

run

`public void` **`run`**`()`

Performs the run process of the Agent and the service registering in the Registry.

Specified by:

`run` in interface `IAgent`

---

stop

`public void` **`stop`**`()`

Performs the stop process of the Agent and the service unregistering in the Registry.

Specified by:

`stop` in interface `IAgent`

unload

```
public void unload()
```

Performs the unload process of the nSOMAgent in the Service Execution Platform.

Specified by:

`unload` in interface `IAgent`

orchestrationListener

```
public void orchestrationListener(int orchestrationId,
                                  String event)
```

Notifies an event to the listener.

Specified by:

`orchestrationListener` in interface `Orchestrator`

Parameters:

`orchestrationId` - Identity of the orchestration assciated to the event.

`event` - Event.

Class OrchestratorEngine

org.upm.nsom.framework.Orchestrator

`java.lang.Object`

`└ org.upm.nsom.frmework.`**`Orchestrator.OrchestratorEngine`**

`public class` **`OrchestratorEngine`**

`extends Object`

The Orchestration Engine provides the mechanism to orchestrate a service in an nSOM environment being in charge of the communications and the execution od the instructions described in the rules provided by the users of this class.

Constructor Summary
`OrchestratorEngine``()` *151*

Method Summary	
`void`	**`addListener`**`(Orchestrator orchestrator, int orchestrationId)` Adds a listener to the events generated by the Orchestrator Engine.
`void`	**`cancel`**`(int orchestrationId)` Cancel an orchestration.
`object`	**`execute`**`(int orchestrationId)` If the orchestrated service is available, it is executed.
`int`	**`parse`**`(String rules)` Parses the rules of orchestration provided and carries out the required communications to compose the service.

Constructor Detail

OrchestratorEngine

`public` **`OrchestratorEngine`**`()`

Method Detail

parse

`public int` **`parse`**`(String rules)`

Parses the rules of orchestration provided and carries out the required communications to compose the service.

Parameters:

`rules` - Rules of orchestration.

Returns:

An identification number for an orchestration.

execute

`public object` **`execute`**`(int orchestrationId)`

If the orchestrated service is available, it is executed.

Parameters:

`orchestrationId` - Identity of the orchestration to be executed.

Returns:

The response of the orchestration, if any.

Throws:

`OrchestrationException` - in case there is an error while performin the orchestration.

`OrchestrationNotAvailable` - in case the orchestration identified by orchestrationId is not available.

`OrchestrationUnknown` - if the orchestrationId does not correspond to any orhcestration registered in the engine.

---

cancel

```
public void cancel(int orchestrationId)
```

Cancel an orchestration.

Parameters:

`orchestrationId` - The identity of the orchestration to be canceled.

---

addListener

```
public void addListener(Orchestrator orchestrator,
                        int orchestrationId)
```

Adds a listener to the events generated by the Orchestrator Engine.

Parameters:

`orchestrator` - The Orchestrator that will receive the notifications.

`orchestrationId` - The orchestration that the agent wants to be notified for.

Anexo III

API de servicios REST

Todas las URI empleadas en esta API tienen como base de referencia la URI "/cxf/".

API REST de servicio de registro

Recurso	Método	Descripción
wooreg	POST	Crea una nueva entrada de servicio y/o de dispositivo en el Registro de Servicios y Dispositivos
	GET	Recupera la información de todos los dispositivos almacenados en el Registro y sus correspondientes servicios asociados.
wooreg?deviceID={deviceID}	GET	Recupera la información de los servicios asociados al dispositivo cuya identidad coincida con *deviceID.*
wooreg?deviceType={type}	GET	Recupera la información de los dispositivos que correspondan al tipo *type* y lo servicios asociados.
wooreg/{deviceID}	DELETE	Elimina del registro toda la información asociada al dispositivo identificado por *deviceID*.

API REST de acceso a servicios

Recurso	Método	Descripción
{deviceID}/{servicio}	GET	Invoca al *servicio* ofrecido por el dispositivo cuya identidad es *deviceID*.

Anexo IV

Comparativa de tecnologías de red

Tecnología	Red	Topología	Energía	Velocidad	Rango
NFC	PAN	P2P	Muy baja	400 Kbps	< 10 cm
RFID	PAN	P2P	Muy baja	400 Kbps	< 3 m
Bluetooth	PAN	Estrella	Baja	700 Kbps	< 20 m
Bluetooth LE[1]	PAN	Estrella	Muy baja	1 Mpbs	5 – 10 m
ANT	PAN	P2P, Estrella, Árbol, Malla	Muy baja	1 Mbps	1 – 30 m
Wi-Fi®	LAN	Estrella	Baja – Alta	11 – 100 Mbps	4 – 20 m
ZigBee®	LAN	Malla, Estrella, Árbol	Muy baja	250 Kbps	10-300 m
Z-wave	LAN	Malla	Muy baja	40 Kbps	30 m
KNX	LAN	Malla, Estrella, Árbol	Muy baja	1,2 Kbps	800 m
Wireless HART	LAN	Malla, Estrella	Muy baja	250 Kbps	200 m
6LoWPAN	LAN	Malla, Estrella	Muy baja	250 Kbps	800 m
WiMAX	MAN	Malla	Alta	11 – 100 Mbps	50 km
3.5 G	WAN	Malla	Alta	1,8 – 7,2 Mbps	Celular

[1] Bluetooth para baja energía: http://www.bluetooth.com/Pages/low-energy.aspx

Anexo V

Comparativa de productos ESB

Nombre	Fabricante	Precio	Licencia	Sistemas operativos				Tecnologías soportadas
				Win	Linux	OS X	Otros	
WebSphere ESB	IBM	$39,400	Propietaria	SÍ	SÍ	NO	SÍ	MQ, JMS, EJB, SOAP, REST
BizTalk	Microsoft	$620 - $10,835	Propietaria	SÍ	NO	NO	NO	AS1, AS2, AS3, LDAP, XMPP, SFTP
Apache ServiceMix	Apache	Gratuito	Apache	SÍ	SÍ	SÍ	SÍ	JBI, MQ, OSGi, Spring
Fuse ESB Enterprise[1][2]	Red Hat	Gratuito	Basada en Apache	SÍ	SÍ	SÍ	SÍ	JBI, MQ, OSGi, Spring, SOAP, REST, HTTP, HTTPS, XMPP, POJO
Mule ESB[2]	Mule Soft	Gratuito	CPAL	SÍ	SÍ	SÍ	—	JBI, MQ, OSGi, Spring, SOAP, REST
Open ESB	Open ESB Community	Gratuito	CDDL	SÍ	SÍ	SÍ	SÍ	JBI, HTTP, SOAP, JMS, LDAP, REST, HL7, BPEL, POJO, IEP
Petals ESB	OW2 Consortium	Gratuito	LGPL 2.1	SÍ	SÍ	SÍ	SÍ	JBI, SOAP, FTP, HTTP, JMS, JDBC, SMTP/POP/IMAP, EJB, BPEL, POJO
UltraESB	AdroitLogic	Gratuito	Adroit Logic EULA	SÍ	SÍ	SÍ	SÍ	AS2, MLLP/S, HL7, XACML, JTA XA, JMS, AMQ, REST, SOAP
Talend ESB	Talend	Subscripción	Dual	SÍ	SÍ	SÍ	SÍ	—
ActiveMatrix Service Bus	TIBCO	Variable	Propietaria	SÍ	SÍ	NO	SÍ	—
WSO2 ESB	WSO2	Gratuito	Apache	SÍ	SÍ	—	SÍ	AMQP, SOAP, REST, FIX, POJO, JMS
webMethods IS	Software AG	Variable	Propietaria	Sí	—	—	—	JMS, SOAP, LDAP, SMTP, SAP, Siebel
Oracle Service Bus	Oracle		OTN	Sí	—	—	—	AQ, MQ, JMS, SOAP, WSIF, OA
ESB Suite	Adeptia	Variable	Propietaria	—	—	—	—	—

[1] Actualmente conocido como Red Hat jBoss Fuse
[2] Basado en Apache ServiceMix

GLOSARIO

TÉRMINO	DEFINICIÓN	FUENTE
Actuador	Un actuador es un objeto que realiza acciones. La actuación es el mecanismo por el que una aplicación puede actuar sobre el entorno. La decisión de activar el actuador puede provenir de cualquier otro elemento de la red.	[80]
Architecture	La organización fundamental de un sistema representada por sus componentes, las relaciones entre ellos y con el entorno, y los principios que guían su diseño y evolución.	[81]
Dispositivo	Componente técnico físico (hardware) con capacidades de comunicación hacia otros sistemas TI. Un *dispositivo* puede estar asociado o ser parte interna de una *Entidad Física*, o monitorizar una *Entidad Física* en su vecindad.	[27]
Entidad física	Cualquier objeto físico que es relevante desde la perspectiva de un usuario o de una aplicación.	[27]
Framework	Un *framework* (marco de trabajo) es un diseño reutilizable de un sistema o parte de él que está representado por un conjunto de clases abstractas y la forma en la que interactúan sus instancias.	[82]
Gateway	Una puerta de enlace (Gateway) es un elemento de reenvío, permitiendo la conexión de varias redes locales. Un Gateway puede ser implementado en un *dispositivo* que provea la traducción de protocolos entre las ramas periféricas de IoT que sean provistas con los niveles inferiores de las pilas de comunicación. Por cuestiones de eficiencia las puertas de enlace pueden actuar en diferentes niveles, dependiendo de cuál sea el nivel inferior en una implementación de protocolo común. Las puertas de enlace pueden además proveer de soporte para la seguridad, la escalabilidad, el descubrimiento de servicios, la geolocalización, la tarificación, etc.	[27]
Interfaz	Conjunto nominal de operaciones que caracterizan el comportamiento de una entidad	

Internet de las Cosas (IoT)	La red global que conecta cualquier objeto inteligente.	[27]
Interoperabilidad	La capacidad de compartir información y servicios. La capacidad de dos o más sistemas o componentes para intercambiar y usar información. La capacidad de los sistemas para proveer y recibir servicios de otros sistemas y usar los servicios así intercambiados para permitirles operar juntos con eficacia.	
M2M	Comunicación automática entre *dispositivos* sin intervención humana. A menudo se hace referencia a un sistema de sensores remotos que están transmitiendo datos de forma continua a un sistema central. Algunos ejemplos son los sistemas de detección climatológica para agricultura, los contadores automáticos, y las etiquetas RFID.	
Middleware	Se define comúnmente como una capa de software entre las aplicaciones y los sistemas operativos, facilitando a los programadores de aplicaciones un mayor nivel de abstracción, como por ejemplo la invocación de procedimientos remotos, el intercambio confiable de mensajes o las transacciones. Estas abstracciones simplifican considerablemente la construcción de sistemas distribuidos, y como resultado los productos *middleware* se están adoptando con rapidez por la industria [83], siendo percibida como una tecnología de éxito.	[73]
Modelo de Referencia	Un modelo de referencia es un marco de trabajo abstracto para la comprensión de las relaciones significativas entre las entidades de algún entorno. Con él se facilita el desarrollo de referencias específicas o arquitecturas concretas empleando normativas o especificaciones consistentes que soporten dicho entorno. Un modelo de referencia se compone de un conjunto mínimo de conceptos, axiomas y relaciones dentro del dominio de un determinado problema, y es independiente de normas específicas,	[28]

	tecnologías, implementaciones o detalles concretos.	
Red Inalámbrica de Sensores y Actuadores (WSAN)	Las Redes Inalámbricas de Sensores y Actuadores (WSAN) son redes de nodos que perciben y, potencialmente, controlan el entorno. Comunican la información mediante enlaces inalámbricos permitiendo la interacción entre personas u ordenadores y el entorno que les rodea.	
Red No Restringida (NTU)	Las redes no restringidas (NTU por sus siglas en inglés) están caracterizadas por enlaces de comunicación de alta velocidad (ofreciendo ratios de transferencia en el orden de los Mbit/s o superiores), tales como la Internet cableada actual. Las latencias en el nivel de enlace son cortas, y principalmente afectadas por una posible congestión de red que por la propia tecnología física de transmisión.	[27]
Red Restringida (NTC)	Las redes restringidas (NTC por sus siglas en inglés) están caracterizadas por ratios de transferencia bajos, típicamente inferiores a 1 Mbit/s, como los ofrecidos por IEEE 802.15.4. Estas redes también se caracterizan por sus altas latencias debidas a múltiples factores que entre otros incluyen: 1. La tecnología de baja velocidad a nivel físico. 2. Las políticas de ahorro de energía en los terminales que pueblan este tipo de redes, que pueden implicar el apagado periódico de sus sistemas de radio con el propósito de mejorar su eficiencia energética.	[27]
Redes de Próxima Generación (NGN)	Red de paquetes capaz de proveer servicios de telecomunicación y capaz de emplear múltiples anchos de banda, tecnologías de transporte provistas de QoS y en las que las funciones relacionadas con el servicio son independientes de las tecnologías relacionadas con el transporte	[84]
Sensor	*Dispositivo* especial que obtiene medidas de las características físicas de una o más *Entidades físicas*.	[27]

REFERENCIAS BIBLIOGRÁFICAS

[1] **GIFFINGER, Rudolf… [et al.].** *Smart cities — Ranking of European medium-sized cities.* Centre of Regional Science, Viena University of Technology. 2007. Disponible en Internet: http://www.smart-cities.eu/download/smart_cities_final_report.pdf.

[2] **European Commission.** *EUROPE 2020: A strategy for smart, sustainable and inclusive growth.* Bruselas : European Commission, 2010. Disponible en Internet: http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=COM:2010:2020:FIN:EN:PDF.

[3] **European Commission**. Energy: Smart Cities and Communities - European Commission. [En línea] European Commission. [Citado el: 30 de 07 de 2013.] http://ec.europa.eu/energy/technology/initiatives/smart_cities_en.htm.

[4] **European Commission**. *Smart Cities and Communities — European Innovation Partnership.* Brussels : European Commission, 2012. Disponible en Internet: http://ec.europa.eu/energy/technology/initiatives/doc/2012_4701_smart_cities_en.pdf.

[5] **European Commission**. European Initiative on Smart Cities. *European Initiative on Smart Cities.* [En línea] Strategic Energy Technologies Information System. [Citado el: 10 de 8 de 2013.] http://setis.ec.europa.eu/implementation/technology-roadmap/european-initiative-on-smart-cities.

[6] **European Commission.** Pacto de los Alcaldes. [En línea] [Citado el: 30 de 7 de 2013.] http://www.pactodelosalcaldes.eu/.

[7] **ASHTON, Kevin.** That 'Internet of Things' Thing. [En línea] 22 de 6 de 2009. [Citado el: 31 de 7 de 2013.] http://www.rfidjournal.com/articles/view?4986.

[8] **ITU.** *The Internet of Things: Executive Summary.* ITU. s.l. : ITU, 2005. Disponible en Internet: http://www.itu.int/dms_pub/itu-s/opb/pol/S-POL-IR.IT-2005-SUM-PDF-E.pdf.

[9] **Beecham Research.** M2M/IoT Sector Map :: Beecham Research. [En línea] Beecham Research. [Citado el: 30 de 8 de 2013.] http://www.beechamresearch.com/article.aspx?id=4.

[10] Web of Objects. [En línea] [Citado el: 30 de 7 de 2013.] http://www.web-of-objects.com/.

[11] **ITEA 2.** Itea 2 — Web of Objects. [En línea] [Citado el: 30 de 7 de 2013.] http://www.itea2.org/project/index/view?project=10097.

[12] **GATELLIER, Patrick.** *Smart Objects for a Smarter City.* s.l. : ITEA2, 2012. Disponible en Internet: http://www.itea2.org/project/result/download?result=6439&file=10028_Web_of_Objects_Project_Leaflet_WoO_pr_oct_12.pdf.

[13] **GATELLIER, Patrick**. *Project poster for ITEA2 and Artemis Co-Summit 2012.* 2012. Disponible en Internet: http://www.itea2.org/project/workpackage-document/download?document=893&file=10028_Web_of_Objects_WP7_WoO_project_poster_for_ITEA_2_and_Artemis_Co_Summit_2012.pdf.

[14] **GATELLIER, Patrick**. *Web of Objects: Full Project Proposal.* ITEA2. 2012.

[15] **GARCÍA HERNANDO, Ana Belén… [et al.].** *Problem Solving for Wireless Sensor Networks.* s.l. : Springer, 2008. ISBN: 978-1-84800-202-9.

[16] **SOHRABY, Kazem; MINOLI, Daniel; ZNATI, Taieb.** *Wireless Sensor Networks: Technology, Protocols and Applications.* Hoboken, New Jersey : John Wiley & Sons, Inc., 2007. ISBN 978-0-471-74300-2.

[17] **HOLGER, Karl; WILLIG, Andreas.** *Protocols and Architectures for Wireless Sensor Networks.* s.l. : John Wiley & Sons, Ltd., 2005. ISBN-10 0-470-09510-5.

[18] **ETSI.** *Machine-to-Machine communications (M2M): M2M service requirements.* ETSI TS 102 689 V2.1.1. Sophia Antipolis Cedex : ETSI, 2013. Disponible en Internet: http://www.etsi.org/deliver/etsi_ts/102600_102699/102689/02.01.01_60/ts_102689v020101p.pdf.

[19] **LOVETT, Mark.** Machine-to-Machine Technology = Efficient Economy. *Trenton Systems Blog.* [En línea] 2011. [Citado el: 30 de 8 de 2013.] http://blog.trentonsystems.com/machine-to-machine-technology-efficient-economy/.

[20] **ETSI.** Machine to Machine Communications. [En línea] ETSI, 2012. [Citado el: 1 de 8 de 2013.] http://www.etsi.org/technologies-clusters/technologies/m2m.

[21] **ETSI**. Machine to Machine Communications. [aut. libro] GSM Association. *Mobile World Congress (2011).* Barcelona : ETSI, 2011. Disponible en Internet: http://www.etsi.org/WebSite/document/EVENTS/ETSI M2M Presentation during MWC 2011.pdf.

[22] **ETSI**. *Machine-to-Machine communications (M2M); Functional architecture.* ETSI TS 102 690 V1.2.1. Sophia Antipolis Cedex : ETSI, 2013. Disponible en Internet: http://www.etsi.org/deliver/etsi_ts/102600_102699/102690/01.02.01_60/ts_102690v010201p.pdf.

[23] **ETSI**. *Machine-to-Machine communications (M2M): mIa, dIa and mId interfaces.* ETSI TS 102 921. Sophia Antipolis Cedex : ETSI, 2012. Disponible en Internet: http://www.etsi.org/deliver/etsi_ts/102900_102999/102921/01.02.01_60/ts_102921v010201p.pdf.

[24] **IoT-I.** The Internet of Things Initiative. [En línea] 2013. [Citado el: 1 de 8 de 2013.] http://www.iot-i.eu/public.

[25] **IoT-A.** Internet of Things - Architecture. [En línea] 2013. [Citado el: 1 de 8 de 2013.] http://www.iot-a.eu/.

[26] **IoT-I.** IoT-Pedia. [En línea] 2013. [Citado el: 1 de 8 de 2013.] http://www.iot-pedia.org.

[27] **MAGERKURTH, Carsten… [et al.].** *D1.4 Converged architectural reference model for the IoT v2.0.* [ed.] Carsten Magerkurth. 2012. Disponible en Internet: http://www.iot-a.eu/public/public-documents/documents-1/1/1/D1.4/at_download/file.

[28] **MACKENZIE, C. Matthew… [et al.].** *Reference Model for Service Oriented Architecture 1.0.* s.l. : OASIS, 2006. Disponible en Internet: http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.pdf.

[29] **MARTIN, Gregorio… [et al.].** *D2.1 — Rource Description Specification.* s.l. : IoT-A, 2012. Disponible en Internet: http://www.iot-a.eu/public/public-documents/documents-1/1/1/copy3_of_d1.2/at_download/file.

[30] **ISO.** *Information technology — Open Systems Interconnection — Basic Reference Model: The Basic Model.* 1996. ISO/IEC 7498-1:1994.

[31] **CUERVA GARCÍA, Alexandra.** *Redes y servicios ubicuos para Internet de las Cosas en dispositivos llevables.* Proyecto Fin de Carrera. Universidad Politécnica de Madrid. Escuela Universitaria de Ingeniería Técnica de Telecomunicación, 2012. Disponible en Internet: http://oa.upm.es/13950/.

[32] **GRUSCHKA, Nils… [et al.].** *D4.2 - Concepts and Solutions for Privacy and Security in the Resolution Infrastructure.* s.l. : IoT-A, 2012. Disponible en Internet: http://www.iot-a.eu/public/public-documents/documents-1/1/1/d4.2/at_download/file.

[33] **GAMBETTA, Diego.** *Trust: Making and Breaking Cooperative Relations.* s.l. : Basil Blackwell Ltd., 1988. Disponible en Internet: http://www.nuffield.ox.ac.uk/users/gambetta/Trust_making and breaking cooperative relations.pdf. ISBN: 0-631-15506-6.

[34] **WOODS, Eoin y ROZANSKI, Nick.** *Using Architectural Perspectives.* s.l. : IEEE, 2005, 5th Working IEEE/IFIP Conference on Software Architecture (WICSA'05), págs. 25-35.

[35] **ZENG, Deze; GUO, Song; CHENG, Zixue.** *The Web of Things: a Survey (invited paper).* 6, 2011, Journal of Communication, Vol. 6, págs. 424-438.

[36] **AKRIBOPOULOS, Orestis… [et al.].** *A Web Services-oriented Architecture for Integrating Small Programmable Objects in the Web of Things.* [ed.] IEEE. London : s.n., 2010, Developments in E-systems Engineering, págs. 70-75. ISBN: 978-1-4244-8044-9.

[37] **ROSSI, Michele, … [et al.].** *D3.3 Initial IoT Protocol Suite definition.* [ed.] Michele Rossi. 2012. Disponible en Internet: http://www.iot-a.eu/public/public-documents/documents-1/1/1/d3.3/at_download/file.

[38] **ZigBee Alliance.** ZigBee Alliance. [En línea] ZigBee Alliance, 2013. [Citado el: 10 de 8 de 2013.] http://www.zigbee.org.

[39] **MONTENEGRO, G … [et al.].** *Transmission of IPv6 Packets over IEEE 802.15.4 Networks.* s.l. : IETF, 2007. Disponible en Internet: http://www.ietf.org/rfc/rfc4944. RFC 4944.

[40] **KUSHALNAGAR, N; MONTENEGRO, G; SCHUMACHER, C.** *IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals.* s.l. : IETF, 2007. Disponible en Internet: http://tools.ietf.org/rfc/rfc4919. RFC 4919.

[41] **HUI, J; THUBERT, P.** *Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks.* s.l. : IETF, 2011. Disponible en Internet: http://tools.ietf.org/rfc/rfc6282. RFC 6282.

[42] **DEERING, S; HINDEN, R.** *Internet Protocol, Version 6 (IPv6).* s.l. : IETF, 1998. Disponible en Internet: http://tools.ietf.org/rfc/rfc2460. RFC2460.

[43] **FIELDING, Roy… [et al.].** *HyperText Transfer Protocol - HTTP/1.1.* s.l. : IETF, 1999. Disponible en Internet: http://tools.ietf.org/html/rfc2616. RFC 2616.

[44] **DUSSEAULT, L; SNELL, J.** *PATCH Method for HTTP.* s.l. : IETF, 2010. Disponible en Internet: http://tools.ietf.org/html/rfc5789. RFC 5789; ISSN: 2070-1721.

[45] **SHELBY, Z … [et al.].** *Constrained Application Protocol.* s.l. : IETF, 2012. Disponible en Internet: http://tools.ietf.org/html/draft-ietf-core-coap-13.

[46] Constrained RESTful Environment. [En línea] IETF. [Citado el: 10 de 8 de 2013.] https://datatracker.ietf.org/wg/core/.

[47] **BRAY, Tim… [et al.].** *Extensible Markup Language (XML) 1.1. (Second Edition).* s.l. : W3C, 2006. Disponible en Internet: http://www.w3.org/TR/xml11/.

[48] **GAO, Shudi; SPERBERG-MCQUEEN, C.M.; THOMPSON, Henry S.** *W3C XML Schema Definition Lenguage (XSD) 1.1 Part1: Structures.* s.l. : W3C, 2012. Disponible en Internet: http://www.w3.org/TR/xmlschema11-1/.

[49] **SCHNEIDER, John; KAMIYA, Takuki.** *Efficient XML Interchange (EXI) Format 1.0.* s.l. : W3C, 2011. Disponible en Internet: http://www.w3.org/TR/exi.

[50] **CROCKFORD, Douglas.** *The application/json Media Type for JavaScript Object Notation.* s.l. : IETF, 7 de 2006. Disponible en Internet: http://tools.ietf.org/html/rfc4627. RFC 4627.

[51] **ZYP, Kris; GALIEGUE, Francis; COURT, Gary.** *JSON Schema: core definitions and terminology.* s.l. : IETF, 31 de 1 de 2013. Disponible en Internet: http://tools.ietf.org/html/draft-zyp-json-schema-04. draft-zyp-json-schema-04.

[52] **CHINNICI, Roberto… [et al.].** *Web Services Description Language (WSDL) Versión 2.0.* s.l. : W3C, 2007. Disponible en Internet: http://www.w3.org/TR/wsdl20/.

[53] **HADLEY, Marc.** *Web Application Description Language.* s.l. : Sun Microsystems, Ltd., 2013. Disponible en Internet: http://www.w3.org/Submission/wadl/.

[54] Service Mapping Description Proposal. [En línea] [Citado el: 10 de 8 de 2013.] https://dojotoolkit.org/reference-guide/1.9/dojox/rpc/smd.html.

[55] **GUDGIN, Martin… [et al.].** *Simple Object Access Protocol.* s.l. : W3C, 2007. Disponible en Internet: http://www.w3.org/TR/soap12-part1/.

[56] **FIELDING, Roy Thomas.** *Architectural Styles and the Design of Network-based Software Architectures.* Irvine : University of California, 2000. Disponible en Internet: http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm.

[57] **OSGi Alliance.** OSGi Technology. [En línea] OSGi Alliance. [Citado el: 10 de 8 de 2013.] http://www.osgi.org/Technology/HomePage.

[58] **PAPAZOGLOU, Michael P. … [et al.].** *Service-Oriented Computing: State of the Art and Research Challenges.* 11 de 2007, IEEE Computer, Vol. 40, págs. 38-35.

[59] **ERL, Thomas.** Service-Oriented Architecture: Concepts, Technology, and Design. [En línea] [Citado el: 12 de 8 de 2013.] http://soaprinciples.com/.

[60] **SCHULTE, Roy W.** *Predicts 2004: Enterprise Service Buses Are Taking Off.* s.l. : Gartner, 2003.

[61] **CHAPPELL, David.** *Enterprise Service Bus.* s.l. : O'Reilly Media, 2004. ISBN-10: 0596006756.

[62] **Red Hat, Inc.** *Fuse ESB Enterprise: Product Introduction.* s.l. : Red Hat, Inc., 2013. Disponible en Internet: https://access.redhat.com/site/documentation/en-US/Fuse_ESB_Enterprise/7.1/pdf/Product_Introduction/Fuse_ESB_Enterprise-7.1-Product_Introduction-en-US.pdf.

[63] **TEN-HOVE, Ron; WALKER, Peter.** JSR 208: JavaTM Business Integration (JBI). [En línea] Sun Microsystems, Inc., 25 de 8 de 2005. [Citado el: 12 de 8 de 2013.] http://jcp.org/en/jsr/detail?id=208.

[64] **Red Hat, Inc.** *Fuse ESB Enterprise Data Sheet.* [PDF] Red hat, Inc. : s.n., 2012. Disponible en Internet: http://fusesource.com/collateral/download/172.

[65] **POSLAD, Stefan.** *Ubiquitous Computing: Smart Devices, Environments and Interactions.* West Sussex : John Wiley & Sons Ltd, 2009. ISBN 978-0-470-03560-3.

[66] WebServices, Orchestration / choreography. [En línea] 27 de 3 de 2012. [Citado el: 20 de 8 de 2013.] http://jcastellssala.wordpress.com/2012/03/27/webservices-orchestration-choreography/.

[67] **Object Management Group (OMG).** *Business PRocess Model and Notation (BPMN).* s.l. : Object Management Group (OMG), 2011. Disponible en Internet: http://www.omg.org/spec/BPMN/2.0.

[68] **JORDAN, Diane… [et al.].** *Web Services Business Process Execution Language Version 2.0.* s.l. : OASIS, 2007. Disponible en Internet: http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html.

[69] WS-BPEL Runtime User Guide. *Hello World Example.* [En línea] jBOSS jBPM. [Citado el: 25 de 8 de 2013.] http://docs.jboss.com/jbpm/bpel/v1.1/userguide/tutorial.hello.html.

[70] **KANATZAS, Nickolas… [et al.].** *Web Services Choreography Description Language Version 1.0.* s.l. : W3C, 2005. Disponible en Internet: http://www.w3.org/TR/ws-cdl-10/.

[71] **MISRA, Jayadev; HOARE, Tony; MENZEL, Galen.** *A Tree Semantics of an Orchestration Language.* Marktoberdorf : s.n., 8 de 2004, Lecture Notes for NATO summer school. Disponible en línea: http://orc.csres.utexas.edu/papers/Semantics.Orc.pdf.

[72] Orc Language Project. [En línea] 2012. [Citado el: 12 de 8 de 2013.] http://orc.csres.utexas.edu/index.shtml.

[73] **EMMERICH, Wolfgang; AOYAMA, Mikio; SVENTEK, Joe.** *The impact of research on middleware technology.* ACM, Enero de 2007, Vol. 41, págs. 89-112. Disponible en Internet: http://dl.acm.org/citation.cfm?id=1228310. ISSN 0163-5980.

[74] **FAMILIAR, Miguel S.; MARTÍNEZ, José F.; LÓPEZ, Lourdes.** *Pervasive Smart Spaces and Environments: A Service-Oriented Middleware Architecture for Wireless Ad Hoc and Sensor Networj.* s.l. : Hindawi Publishing Corporation, 27 de 2 de 2012, International Journal of Distributed Sensor Networks, Vol. 2012. Disponible en Internet: http://www.hindawi.com/journals/ijdsn/2012/725190/.

[75] **RODRÍGUEZ MOLINA, Jesús.** *Semantic Middleware Development for the Internet of Things.* Proyecto Fin de Carrera. Universidad Politécnica de Madrid. Escuela Universitaria de Ingeniería Técnica de Telecomunicación, 2012. Disponible en Internet: http://oa.upm.es/13841/.

[76] **MARTÍNEZ ABASCAL, María Jesús.** *Open Technolgies for Prototyping the Internet of Things.* Proyecto Fin de Carrera. Universidad Politécnica de Madrid. Escuela Universitaria de Ingeniería Técnica de Telecomunicación, 2013.

[77] **BISCHOF, Marc… [et al.].** BPELscript - BPEL for Developers. [En línea] [Citado el: 30 de 8 de 2013.] http://www.bpelscript.org/.

[78] **Oracle Corporation.** Sun SPOT World. [En línea] Oracle Corporation, 2010. [Citado el: 1 de 8 de 2013.] http://sunspotworld.com/.

[79] **ASUSTeK Computer Inc.** EeeBox PC EB1033. [En línea] ASUSTeK Computer Inc., 2012. [Citado el: 20 de 8 de 2013.] http://www.asus.com/Eee_Box_PCs/EeeBox_PC_EB1033/.

[80] **ETSI.** *Machine-to-Machine communications (M2M): Definitions.* Sophia Antipolis Cedex : ETSI, 2013. Disponible en Internet: http://www.etsi.org/deliver/etsi_tr/102700_102799/102725/01.01.01_60/tr_102725v010101p.pdf. ETSI TR 102 725 V1.1.1.

[81] **ISO/IEC/IEEE.** *Systems and software engineering: Architecture description.* s.l. : ISO/IEC/IEEE, 2011. ISO/IEC/IEEE 42010.

[82] **JOHNSON, Ralph E.** *Frameworks equal (components + patterns).* 10, s.l. : ACM, 1 de 10 de 1997, Communications of the ACM, Vol. 40, págs. 39-42. ISSN 0001-0782.

[83] **CHARLES, J.** *Middleware Moves to the Forefront.* 5, s.l. : IEEE, Mayo de 1999, Computer, Vol. 32, págs. 17-19. Disponible en Internet: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=762786&isnumber=16523. ISSN 0018-9162.

[84] **ETSI.** *Corporate telecommunication Networks (CN); Mobility for enterprise communication.* s.l. : ETSI, 2006. Disponible en Internet: http://www.etsi.org/deliver/etsi_tr/102400_102499/102477/01.01.01_60/tr_102477v010101p.pdf. TR 102 477.